

AA-N324B-TE



OFFICE MENU

Application Programmer's Reference,
Volume 1: Flow Control

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1985 by Digital Equipment Corporation. All rights reserved.

The postage prepaid Reader's Comments Form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

digital™

ALL-IN-1
DATATRIEVE
DEC
DECmail
DECmate
DECnet
DECspell
DECsystem-10

DECSYSTEM-20
DEctalk
DECUS
DECwriter
DIBOL
MASSBUS
PDP
P/OS
Professional
Rainbow

RSTS
RSX
UNIBUS
VAX
VMS
VT
Work Processor
WPS-PLUS

ALL-IN-1

Guide to ALL-IN-1 Information

For New Users

Computer Based Instruction

ALL-IN-1 Computer Based Instruction (CBI) introduces you to ALL-IN-1 through simple lessons displayed on your terminal. The lessons explain basic ALL-IN-1 features, and take you step by step through some ALL-IN-1 functions. Use the Training (TR) option on the Main Menu and other menus.

Getting Started Guide

The *Getting Started Guide* gives information about basic ALL-IN-1 features. If you are new to ALL-IN-1 or computers, read this book for an introduction and do the simple exercises.

The User's Kit

Help Messages

You can get help anywhere in ALL-IN-1 by pressing the Gold key followed by the H key. ALL-IN-1 explains the choices open to you, and lists related topics on which you can get information.

Quick Lookup

The *Quick Lookup* gives step-by-step instructions for tasks that you can do with ALL-IN-1. It also lists the main keyboard functions and illustrates the special keypad layouts. Use this book along with Help.

User's Reference

The *User's Reference* explains and describes all of the ALL-IN-1 features. Use it as a reference document after you have become familiar with ALL-IN-1 basics.

Installation Guide

The *Installation Guide* tells the VAX/VMS or MicroVMS system manager how to prepare for and install ALL-IN-1. It also describes how to test the success of the installation, set up accounts, and convert ALL-IN-1 Version 1.4 to Version 2.1.

System Manager's Guide

The *System Manager's Guide* gives guidelines and instructions on all aspects of managing ALL-IN-1. It describes management tasks for the user support manager, the ALL-IN-1 manager, and the multi-node manager.

Application Programmer's Reference

The *Application Programmer's Reference* gives a detailed description of the ALL-IN-1 system and the tools available for modifying or adding applications. The three volumes discuss:

- 1 *Flow Control* — The kernel facilities used throughout ALL-IN-1
- 2 *Functions* — The functions that call ALL-IN-1 facilities or routines to perform a specific task
- 3 *Applications* — The integrated applications and how to customize them

Programmer's Mini-Reference

The *Programmer's Mini-Reference* is a quick lookup guide for application programmers. Use it for concise instructions on basic application programming tasks.

The Document Customizer's Kit

Style Guide

The *Style Guide* is the ALL-IN-1 document customizer's handbook. It discusses the ALL-IN-1 documentation strategy, standards, conventions, templates, and tools used in the documentation set. It describes how to find the on-line text files and how to use DIGITAL Standard RUNOFF (DSR). It also gives technical specifications for materials and processes used and packaging information.

Writer's Guide

The *Writer's Guide* discusses how to conduct a technical writing project. It explains how to plan, review, produce, and print documents. Use this book as a guide to documenting ALL-IN-1 or any software documentation project.

Documents Tape

The Documents Tape contains the full text of the *Getting Started Guide*, the *User's Reference*, and the *Quick Lookup* on magnetic tape, in DIGITAL Standard RUNOFF format. The tape also provides command files to help the documenter rebuild the customized documents.

Contents

Volume 1: Flow Control

Preface

1 Overview

- 1.1 Introduction 1-1
- 1.2 ALL-IN-1 in the VAX Information Architecture 1-3
- 1.3 ALL-IN-1 Architecture and Basic Concepts 1-3
 - 1.3.1 Form and Function 1-5
 - 1.3.2 How ALL-IN-1 Is Organized 1-6

2 ALL-IN-1 Initialization and Flow Control

- 2.1 Introduction 2-1
- 2.2 ALL-IN-1 Initialization 2-2
 - 2.2.1 Partial Initialization 2-3
- 2.3 ALL-IN-1 Office Menu Flow Control 2-3

3 Form Processing

- 3.1 Introduction 3-1
- 3.2 Form Libraries 3-2
 - 3.2.1 Searching Form Libraries 3-3

- 3.3 The FORM and SHOW_FORM Functions 3-3
 - 3.3.1 The FORM Function 3-3
 - 3.3.2 The SHOW_FORM Function 3-4
- 3.4 ALL-IN-1 Named Data Conventions 3-4
 - 3.4.1 ALL-IN-1 Form .TYPEs 3-5
 - 3.4.2 The .MORE Directive 3-7
 - 3.4.3 Searching for Named Data 3-7
 - 3.4.4 Form Qualifiers 3-8
 - 3.4.5 Common, or Shared, Form Qualifiers 3-10
- 3.5 The MENU Form 3-24
 - 3.5.1 Menu Named Data 3-25
 - 3.5.2 Menu Form Qualifiers 3-25
 - 3.5.3 Processing Menus 3-28
 - 3.5.4 Stacking Menu Options 3-32
 - 3.5.5 Invoking Menu Options Indirectly 3-33
 - 3.5.6 Creating and Processing Complex Menus 3-33
 - 3.5.7 Menu Stacks and Menu Levels 3-36
- 3.6 The Entry Form 3-37
 - 3.6.1 Entry Form Named Data 3-37
 - 3.6.2 The Data Entry Process: Creating Files 3-38
 - 3.6.3 The Data Entry Process: Accessing Data Files 3-44
 - 3.6.4 Entry Form Qualifiers 3-44
 - 3.6.5 Processing Entry Forms 3-54.1
- 3.7 The Argument Form 3-56
 - 3.7.1 Argument Form Named Data 3-56
 - 3.7.2 Argument Form Qualifiers 3-57
- 3.8 The Calc and Desk Forms 3-57
 - 3.8.1 Calc Form Named Data 3-57
 - 3.8.2 Calc Form Qualifiers 3-58
 - 3.8.3 The Desk Calculator Form 3-58
- 3.9 The Search Form 3-60
 - 3.9.1 Search Form Named Data 3-61
 - 3.9.2 Search Expressions in the .TYPE Directive 3-63
 - 3.9.3 Search Form Qualifiers 3-65
 - 3.9.4 Searching for Strings: The .SFILE Directive 3-65
 - 3.9.5 Formatting Record Collections: The .SOUT Directive 3-67
 - 3.9.6 Search Form Processing 3-68
- 3.10 The Select Form 3-68
 - 3.10.1 Select Form Named Data 3-69
 - 3.10.2 Select Form Qualifiers 3-69
 - 3.10.3 Selecting and Processing Records 3-71
- 3.11 EXIT SCREEN and the PRIOR Form 3-75

- 3.12 Dynamically Redefining Named Data 3-75
 - 3.12.1 The Form .TYPE Parameter in
FORM Function Calls 3-76
 - 3.12.2 The /RESET Form Qualifier 3-77
 - 3.12.3 Dynamically Redefining Entry Forms 3-78
- 3.13 Multipage Form Sets 3-78
- 3.14 Nonstandard Forms 3-80

4 Field Processing

- 4.1 Introduction 4-1
- 4.2 The FIELD Function 4-2
- 4.3 Defining Fields 4-2
- 4.4 Types of Field Qualifiers 4-4
- 4.5 /AUTO 4-5
 - 4.5.1 Examples 4-5
- 4.6 /BLANK 4-5
 - 4.6.1 Example 4-6
- 4.7 /CLEAR 4-6
 - 4.7.1 Examples 4-6
- 4.8 /COMPUTE 4-7
 - 4.8.1 Examples 4-7
- 4.9 /COPY 4-8
 - 4.9.1 Examples 4-9
- 4.10 /DECIMAL 4-10
 - 4.10.1 Examples 4-11
- 4.11 /DOC 4-11
 - 4.11.1 Examples 4-12
- 4.12 /FILE_CHECK 4-14
 - 4.12.1 Examples 4-15
- 4.13 /GET_SAVE 4-16
 - 4.13.1 Examples 4-17
- 4.14 /HARD 4-18
 - 4.14.1 Examples 4-18
- 4.15 /INVALID 4-19
 - 4.15.1 Examples 4-22
- 4.16 /LIST 4-22
 - 4.16.1 Examples 4-23
- 4.17 /OPTIONAL 4-23
 - 4.17.1 Examples 4-24
- 4.18 /POST_FUNCTION 4-24
 - 4.18.1 Examples 4-25
- 4.19 /PRE_FUNCTION 4-25
 - 4.19.1 Examples 4-26

4.20	/PROMPT	4-26
4.20.1	Example	4-27
4.21	/PUT_SAVE	4-27
4.21.1	Examples	4-27
4.22	/RECOGNITION	4-28
4.22.1	Examples	4-30
4.23	/RSE_nnn	4-30
4.23.1	Examples	4-32
4.24	/SCROLL	4-33
4.24.1	/SCROLL with Indexed Files	4-35
4.24.2	SCROLL\$FUNCTION	4-36
4.24.3	Example	4-40
4.24.4	Customized Scrolling	4-43
4.25	/SHOW	4-45
4.25.1	Examples	4-46
4.26	/STORE	4-46
4.27	/UNIQUE	4-47
4.27.1	Example	4-47
4.28	/USE_FORM	4-47
4.29	/VALID	4-47
4.29.1	Examples	4-50
4.30	Data Validation	4-51
4.30.1	Combining Validation Qualifiers	4-54
4.30.2	Controlling Record Collection Output	4-55
4.31	Default Keyboard Definitions	4-56
4.32	Defining the Keyboard	4-63
4.33	Terminator Processing	4-66
4.34	Field Processing	4-69

5 Generic Menu Design

5.1	The Primary Menu Form	5-1
5.1.1	Current Items	5-3
5.1.2	Select (SEL)	5-3
5.1.3	Create (C)	5-4
5.1.4	Edit (E)	5-5
5.1.5	Delete (D)	5-5
5.1.6	Print (P)	5-6
5.1.7	Read (R)	5-6
5.1.8	Index (I)	5-6
5.1.9	Option x (Ox)	5-6
5.1.10	Training (TR)	5-6
5.2	More Menu Options (M)	5-7
5.3	Secondary Menu Form - Additional Menu Options	5-7

Volume 2: Functions

Preface

6 Functions

- 6.1 Introduction 6-1
- 6.2 Function Call Syntax 6-2
- 6.3 Stacking Functions 6-4
 - 6.3.1 Trapping Responses and Error Conditions 6-6
 - 6.3.2 Function Loops 6-7
 - 6.3.3 Forcing Displays 6-7
 - 6.3.4 Other Features 6-7
- 6.4 Pipelining Functions 6-9
- 6.5 User-Defined Functions 6-10
- 6.6 Symbol Substitution in Functions 6-12
- 6.7 ACMSTASK 6-13
 - 6.7.1 Syntax 6-13
 - 6.7.2 Description 6-13
 - 6.7.3 Function Parameters 6-13
 - 6.7.4 Function Qualifiers 6-14
 - 6.7.5 Examples 6-14
 - 6.7.6 Special Features and Restrictions 6-15
- 6.8 ADE 6-16
 - 6.8.1 Syntax 6-16
 - 6.8.2 Description 6-16
 - 6.8.3 Function Parameters 6-16
- 6.9 AI_CHECK 6-17
 - 6.9.1 Syntax 6-17
 - 6.9.2 Description 6-17
 - 6.9.3 Function Parameters 6-17
 - 6.9.4 Function Qualifiers 6-17
 - 6.9.5 Examples 6-18
 - 6.9.6 Special Features and Restrictions 6-18
- 6.10 APPEND 6-19
 - 6.10.1 Syntax 6-19
 - 6.10.2 Description 6-19
 - 6.10.3 Function Parameters 6-19
 - 6.10.4 Examples 6-20
- 6.11 ATTACH 6-21
 - 6.11.1 Syntax 6-21

6.11.2	Description	6-21
6.11.3	Function Parameters	6-21
6.11.4	Function Qualifiers	6-21
6.11.5	Example	6-21
6.11.6	Special Features and Restrictions	6-22
6.12	BYE	6-23
6.12.1	Syntax	6-23
6.12.2	Description	6-23
6.12.3	Function Parameters	6-23
6.12.4	Function Qualifiers	6-23
6.12.5	Examples	6-24
6.12.6	Special Features and Restrictions	6-24
6.13	CABINET	6-25
6.13.1	Syntax	6-25
6.13.2	Description	6-25
6.13.3	Subfunction Descriptions	6-25
6.14	CALENDAR	6-26
6.14.1	Syntax	6-26
6.14.2	Description	6-26
6.14.3	Subfunction Descriptions	6-26
6.14A	CHECK_DISKQUOTA	6-26.1
6.14A.1	Syntax	6-26.1
6.14A.2	Description	6-26.1
6.14A.3	Function Parameters	6-26.1
6.14A.4	Function Qualifiers	6-26.2
6.14A.5	Examples	6-26.2
6.14A.6	Special Features and Restrictions	6-26.2
6.15	CLEAR	6-27
6.15.1	Syntax	6-27
6.15.2	Description	6-27
6.15.3	Function Parameters	6-27
6.15.4	Function Qualifiers	6-27
6.15.5	Examples	6-28
6.15.6	Special Features and Restrictions	6-28
6.16	CLOSE_PRIOR	6-29
6.16.1	Syntax	6-29
6.16.2	Description	6-29
6.16.3	Function Parameters	6-29
6.16.4	Function Qualifiers	6-29
6.16.5	Examples	6-30
6.17	COMMAND	6-31
6.17.1	Syntax	6-31
6.17.2	Description	6-31
6.17.3	Function Parameters	6-31

6.17.4	Function Qualifiers	6-32
6.17.5	Examples	6-33
6.17.6	Special Features and Restrictions	6-34
6.18	COMPUTE	6-37
6.18.1	Syntax	6-37
6.18.2	Description	6-37
6.18.3	Function Parameters	6-37
6.18.4	Function Qualifiers	6-39
6.18.5	Examples	6-39
6.18.6	Special Features and Restrictions	6-41
6.19	CONTROL_C	6-42
6.19.1	Syntax	6-42
6.19.2	Description	6-42
6.19.3	Function Parameters	6-42
6.20	COPY	6-43
6.20.1	Syntax	6-43
6.20.2	Description	6-43
6.20.3	Function Parameters	6-43
6.20.4	Function Qualifiers	6-44
6.20.5	Examples	6-44
6.20.6	Special Features and Restrictions	6-44
6.21	CREATE	6-45
6.21.1	Syntax	6-45
6.21.2	Description	6-45
6.21.3	Function Parameters	6-45
6.21.4	Function Qualifiers	6-46
6.21.5	Examples	6-46
6.21.6	Special Features and Restrictions	6-46
6.22	CXP	6-47
6.22.1	Syntax	6-47
6.22.2	Description	6-47
6.22.3	Subfunction Descriptions	6-47
6.23	DATE_CONVERT	6-48
6.23.1	Syntax	6-48
6.23.2	Description	6-48
6.23.3	Function Parameters	6-48
6.23.4	Function Qualifiers	6-49
6.23.5	Examples	6-50
6.24	DCL	6-51
6.24.1	Syntax	6-51
6.24.2	Description	6-51
6.24.3	Function Parameters	6-51
6.24.4	Function Qualifiers	6-52
6.24.5	Examples	6-52
6.24.6	Special Features and Restrictions	6-52

6.25	DECIMAL	6-54
6.25.1	Syntax	6-54
6.25.2	Description	6-54
6.25.3	Function Parameters	6-54
6.25.4	Function Qualifiers	6-55
6.25.5	Examples	6-55
6.25.6	Special Features and Restrictions	6-55
6.26	DELETE_FILE	6-56
6.26.1	Syntax	6-56
6.26.2	Description	6-56
6.26.3	Function Parameters	6-56
6.26.4	Function Qualifiers	6-57
6.26.5	Examples	6-57
6.26.6	Special Features and Restrictions	6-57
6.27	DESTINATION	6-58
6.27.1	Syntax	6-58
6.27.2	Description	6-58
6.27.3	Function Parameters	6-58
6.27.4	Function Qualifiers	6-58
6.27.5	Special Features and Restrictions	6-58
6.28	DISPLAY	6-59
6.28.1	Syntax	6-59
6.28.2	Description	6-59
6.28.3	Function Parameters	6-59
6.28.4	Function Qualifiers	6-60
6.28.5	Examples	6-60
6.28.6	Special Features and Restrictions	6-60
6.29	DO	6-63
6.29.1	Syntax	6-63
6.29.2	Description	6-63
6.29.3	Function Parameters	6-63
6.29.4	Function Qualifiers	6-64
6.29.5	Examples	6-64
6.29.6	Special Features and Restrictions	6-64
6.30	DTR	6-65
6.30.1	Syntax	6-65
6.30.2	Description	6-65
6.30.3	Function Parameters	6-65
6.30.4	Function Qualifiers	6-66
6.30.5	Examples	6-66
6.30.6	Special Features and Restrictions	6-67

6.31	DUMP_CACHE	6-69
6.31.1	Syntax	6-69
6.31.2	Description	6-69
6.31.3	Function Parameters	6-69
6.31.4	Function Qualifiers	6-69
6.31.5	Examples	6-70
6.31.6	Special Features and Restrictions	6-70
6.32	EDIT	6-71
6.32.1	Syntax	6-71
6.32.2	Description	6-71
6.32.3	Function Parameters	6-71
6.32.4	Function Qualifiers	6-73
6.32.5	Special Features and Restrictions	6-73
6.33	EDT	6-74
6.33.1	Syntax	6-74
6.33.2	Description	6-74
6.33.3	Function Parameters	6-74
6.33.4	Function Qualifiers	6-74
6.33.5	Special Features and Restrictions	6-75
6.34	ERROR	6-76
6.34.1	Syntax	6-76
6.34.2	Description	6-76
6.34.3	Function Parameters	6-76
6.34.4	Function Qualifiers	6-76
6.34.5	Examples	6-77
6.34.6	Special Features and Restrictions	6-77
6.35	EXIT	6-78
6.35.1	Syntax	6-78
6.35.2	Description	6-78
6.35.3	Function Parameters	6-78
6.35.4	Function Qualifiers	6-78
6.35.5	Examples	6-79
6.35.6	Special Features and Restrictions	6-79
6.36	FHELP	6-80
6.36.1	Syntax	6-80
6.36.2	Description	6-80
6.36.3	Function Parameters	6-80
6.36.4	Function Qualifiers	6-80
6.36.5	Examples	6-80

6.37	FIELD	6-81
6.37.1	Syntax	6-81
6.37.2	Description	6-81
6.37.3	Function Parameters	6-81
6.37.4	Function Qualifiers	6-82
6.37.5	Examples	6-82
6.37.6	Special Features and Restrictions	6-82
6.38	FOR	6-83
6.38.1	Syntax	6-83
6.38.2	Description	6-83
6.38.3	Function Parameters	6-83
6.38.4	Boolean Expressions	6-85
6.38.5	Special DO Subfunctions	6-87
6.38.6	Examples	6-88
6.38.7	Special Features and Restrictions	6-90
6.39	FORCE	6-92
6.39.1	Syntax	6-92
6.39.2	Description	6-92
6.39.3	Function Parameters	6-92
6.39.4	Function Qualifiers	6-92
6.39.5	Examples	6-93
6.40	FORM	6-94
6.40.1	Syntax	6-94
6.40.2	Description	6-94
6.40.3	Function Parameters	6-94
6.40.4	Form Qualifiers	6-96
6.41	FORMAT	6-97
6.41.1	Syntax	6-97
6.41.2	Description	6-97
6.41.3	Function Parameters	6-97
6.41.4	Function Qualifiers	6-99
6.41.5	Examples	6-99
6.41.6	Special Features and Restrictions	6-100
6.42	GET	6-101
6.42.1	Syntax	6-101
6.42.2	Description	6-101
6.42.3	Function Parameters	6-102
6.42.4	Function Qualifiers	6-103
6.42.5	Examples	6-103
6.42.6	Special Features and Restrictions	6-107

6.43	HELP	6-108
6.43.1	Syntax	6-108
6.43.2	Description	6-108
6.43.3	Function Parameters	6-108
6.43.4	Function Qualifiers	6-109
6.43.5	Examples	6-110
6.43.6	Special Features and Restrictions	6-110
6.44	IFEXIT	6-112
6.44.1	Syntax	6-112
6.44.2	Description	6-112
6.44.3	Function Parameters	6-112
6.44.4	Function Qualifiers	6-112
6.44.5	Examples	6-113
6.44.6	Special Features and Restrictions	6-113
6.45	INCLUDE	6-114
6.45.1	Syntax	6-114
6.45.2	Description	6-114
6.45.3	Function Parameters	6-114
6.45.4	Function Qualifiers	6-114
6.45.5	Examples	6-115
6.45.6	Special Features and Restrictions	6-115
6.46	KEYPAD	6-116
6.46.1	Syntax	6-116
6.46.2	Description	6-116
6.46.3	Function Parameters	6-116
6.46.4	Function Qualifiers	6-116
6.46.5	Examples	6-117
6.46.6	Special Features and Restrictions	6-117
6.47	LIGHTS	6-118
6.47.1	Syntax	6-118
6.47.2	Description	6-118
6.47.3	Function Parameters	6-118
6.47.4	Examples	6-118
6.47.5	Special Features and Restrictions	6-119
6.48	LIST	6-120
6.48.1	Syntax	6-120
6.48.2	Description	6-120
6.48.3	Function Parameters	6-120
6.48.4	Function Qualifiers	6-121
6.48.5	Examples	6-121
6.48.6	Special Features and Restrictions	6-122

6.49	LOG	6-124
6.49.1	Syntax	6-124
6.49.2	Description	6-124
6.49.3	Function Parameters	6-124
6.49.4	Function Qualifiers	6-124
6.49.5	Examples	6-125
6.49.6	Special Features and Restrictions	6-125
6.50	LOGICAL	6-126
6.50.1	Syntax	6-126
6.50.2	Description	6-126
6.50.3	Function Parameters	6-126
6.50.4	Function Qualifiers	6-127
6.50.5	Examples	6-127
6.50.6	Special Features and Restrictions	6-128
6.51	MAIL	6-129
6.51.1	Syntax	6-129
6.51.2	Description	6-129
6.51.3	Subfunction Descriptions	6-129
6.52	MAKE_FILE_NAME	6-130
6.52.1	Syntax	6-130
6.52.2	Description	6-130
6.52.3	Function Parameters	6-130
6.52.4	Function Qualifiers	6-132
6.52.5	Examples	6-132
6.52.6	Special Features and Restrictions	6-133
6.53	MERGE	6-134
6.53.1	Syntax	6-134
6.53.2	Description	6-134
6.53.3	Function Parameters	6-134
6.53.4	Template Directives	6-137
6.53.5	Function Qualifiers	6-139
6.53.6	Examples	6-139
6.53.7	Special Features and Restrictions	6-140
6.54	MERGE_LINE	6-141
6.54.1	Syntax	6-141
6.54.2	Description	6-141
6.54.3	Function Parameters	6-141
6.54.4	Function Qualifiers	6-141
6.54.5	Examples	6-142
6.54.6	Special Features and Restrictions	6-142

6.55	MERGE_LIST	6-143
6.55.1	Syntax	6-143
6.55.2	Description	6-143
6.55.3	Function Parameters	6-143
6.55.4	Function Qualifiers	6-145
6.55.5	Examples	6-146
6.55.6	Special Features and Restrictions	6-146
6.56	MODE	6-147
6.56.1	Syntax	6-147
6.56.2	Description	6-147
6.56.3	Function Parameters	6-147
6.56.4	Function Qualifiers	6-148
6.56.5	Examples	6-148
6.56.6	Special Features and Restrictions	6-149
6.57	NEWDIR	6-150
6.57.1	Syntax	6-150
6.57.2	Description	6-150
6.57.3	Function Parameters	6-150
6.57.4	Function Qualifiers	6-151
6.57.5	Examples	6-151
6.57.6	Special Features and Restrictions	6-151
6.58	NEWLIB	6-152
6.58.1	Syntax	6-152
6.58.2	Description	6-152
6.58.3	Function Parameters	6-152
6.58.4	Function Qualifiers	6-153
6.58.5	Examples	6-153
6.58.6	Special Features and Restrictions	6-153
6.59	NEXT_LIST	6-155
6.59.1	Syntax	6-155
6.59.2	Description	6-155
6.59.3	Function Parameters	6-155
6.59.4	Function Qualifiers	6-155
6.59.5	Examples	6-156
6.59.6	Special Features and Restrictions	6-156
6.60	OA\$CLI_GET_SYMBOL	6-157
6.60.1	Syntax	6-157
6.60.2	Description	6-157
6.60.3	Function Parameters	6-157
6.60.4	Function Qualifiers	6-157
6.60.5	Examples	6-158
6.60.6	Special Features and Restrictions	6-158

6.61	OA\$CLI_GET_VALUE	6-159
6.61.1	Syntax	6-159
6.61.2	Description	6-159
6.61.3	Function Parameters	6-159
6.61.4	Function Qualifiers	6-159
6.61.5	Examples	6-160
6.61.6	Special Features and Restrictions	6-160
6.62	OA\$CNV_CONVERT	6-161
6.62.1	Syntax	6-161
6.62.2	Description	6-161
6.62.3	Function Parameters	6-162
6.62.4	Function Qualifiers	6-162
6.62.5	Examples	6-162
6.62.6	Special Features and Restrictions	6-162
6.63	OA\$CNV_DIST_LIST	6-163
6.63.1	Syntax	6-163
6.63.2	Description	6-163
6.63.3	Function Parameters	6-163
6.63.4	Examples	6-164
6.63.5	Special Features and Restrictions	6-164
6.64	OA\$CNV_MERGE	6-165
6.64.1	Syntax	6-165
6.64.2	Description	6-165
6.64.3	Function Parameters	6-166
6.64.4	Function Qualifiers	6-166
6.64.5	Examples	6-166
6.64.6	Special Features and Restrictions	6-166
6.65	OA\$CNV_FROM_SCROLL	6-167
6.65.1	Syntax	6-167
6.65.2	Description	6-167
6.65.3	Function Parameters	6-167
6.65.4	Function Qualifiers	6-167
6.66	OA\$CNV_TO_SCROLL	6-168
6.66.1	Syntax	6-168
6.66.2	Description	6-168
6.66.3	Function Parameters	6-168
6.66.4	Function Qualifiers	6-168
6.67	OA\$DSA_CACHE_STATUS	6-169
6.67.1	Syntax	6-169
6.67.2	Description	6-169
6.67.3	Function Parameters	6-169
6.67.4	Function Qualifiers	6-169
6.67.5	Examples	6-170
6.67.6	Special Features and Restrictions	6-170

6.68	OA\$ENT_SET_xxx	6-171
6.68.1	Syntax	6-171
6.68.2	Description	6-171
6.68.3	Function Parameters	6-171
6.68.4	Function Qualifiers	6-171
6.68.5	Special Features and Restrictions	6-172
6.69	OA\$FBT_INSTALL_LIBRARY	6-173
6.69.1	Syntax	6-173
6.69.2	Description	6-173
6.69.3	Function Parameters	6-173
6.69.4	Function Qualifiers	6-174
6.69.5	Examples	6-174
6.69.6	Special Features and Restrictions	6-174
6.70	OA\$FBT_LIST_LIBTREE	6-175
6.70.1	Syntax	6-175
6.70.2	Description	6-175
6.70.3	Function Parameters	6-175
6.70.4	Function Qualifiers	6-175
6.70.5	Examples	6-176
6.71	OA\$FBT_LIST_TREE	6-177
6.71.1	Syntax	6-177
6.71.2	Description	6-177
6.71.3	Function Parameters	6-177
6.71.4	Function Qualifiers	6-177
6.71.5	Examples	6-178
6.72	OA\$FBT_REMOVE_LIBRARY	6-179
6.72.1	Syntax	6-179
6.72.2	Description	6-179
6.72.3	Function Parameters	6-179
6.72.4	Function Qualifiers	6-179
6.72.5	Examples	6-180
6.72.6	Special Features and Restrictions	6-180
6.73	OA_\$FBT_WRITE_LIBRARY	6-181
6.73.1	Syntax	6-181
6.73.2	Description	6-181
6.73.3	Function Parameters	6-181
6.73.4	Function Qualifiers	6-182
6.73.5	Examples	6-182
6.73.6	Special Features and Restrictions	6-182

6.74	OA\$FLD__nnn	6-183
6.74.1	Syntax	6-183
6.74.2	Description	6-183
6.74.3	Terminator Functions	6-183
6.74.4	Examples	6-186
6.75	OA\$FLD__NEXT__LIST	6-188
6.75.1	Syntax	6-188
6.75.2	Description	6-188
6.75.3	Function Parameters	6-188
6.75.4	Function Qualifiers	6-188
6.76	OA\$FLD__RECOGNITION	6-189
6.76.1	Syntax	6-189
6.76.2	Description	6-189
6.76.3	Function Parameters	6-189
6.76.4	Examples	6-189
6.76.5	Special Features and Restrictions	6-189
6.77	OA\$FLD__SCROLL__OFF	6-190
6.77.1	Syntax	6-190
6.77.2	Description	6-190
6.77.3	Function Parameters	6-190
6.77.4	Examples	6-190
6.78	OA\$FLD__SEARCH	6-191
6.78.1	Syntax	6-191
6.78.2	Description	6-191
6.78.3	Function Parameters	6-191
6.78.4	Examples	6-191
6.78.5	Special Features and Restrictions	6-191
6.79	OA\$FLO__CLEAR__VA	6-192
6.79.1	Syntax	6-192
6.79.2	Description	6-192
6.79.3	Function Parameters	6-192
6.79.4	Function Qualifiers	6-192
6.79.5	Special Features and Restrictions	6-192
6.80	OA\$FLO__CLOSE__LIB	6-193
6.80.1	Syntax	6-193
6.80.2	Description	6-193
6.80.3	Function Parameters	6-193
6.80.4	Function Qualifiers	6-193
6.80.5	Special Features and Restrictions	6-194

- 6.81 OA\$FLO_OPEN_LIB 6-195
 - 6.81.1 Syntax 6-195
 - 6.81.2 Description 6-195
 - 6.81.3 Function Parameters 6-195
 - 6.81.4 Function Qualifiers 6-195
 - 6.81.5 Special Features and Restrictions 6-196
- 6.82 OA\$FRM_SET_FIELD 6-197
 - 6.82.1 Syntax 6-197
 - 6.82.2 Description 6-197
 - 6.82.3 Function Parameters 6-197
 - 6.82.4 Function Qualifiers 6-197
- 6.83 OA\$HAR_CHECK_DECCRT 6-198
 - 6.83.1 Syntax 6-198
 - 6.83.2 Description 6-198
 - 6.83.3 Function Parameters 6-198
 - 6.83.4 Function Qualifiers 6-198
 - 6.83.5 Special Features and Restrictions 6-199
- 6.84 This section has been removed 6-200
- 6.85 OA\$INI_GLOBALS 6-201
 - 6.85.1 Syntax 6-201
 - 6.85.2 Description 6-201
 - 6.85.3 Function Parameters 6-201
 - 6.85.4 Function Qualifiers 6-201
- 6.86 OA\$INI_INITIALIZE 6-202
 - 6.86.1 Syntax 6-202
 - 6.86.2 Description 6-202
 - 6.86.3 Function Parameters 6-202
 - 6.86.4 Function Qualifiers 6-202
 - 6.86.5 Examples 6-203
- 6.87 OA\$INI_LOGICALS 6-204
 - 6.87.1 Syntax 6-204
 - 6.87.2 Description 6-204
 - 6.87.3 Function Parameters 6-204
 - 6.87.4 Function Qualifiers 6-204
- 6.88 OA\$INI_OPEN_LIB 6-205
 - 6.88.1 Syntax 6-205
 - 6.88.2 Description 6-205
 - 6.88.3 Function Parameters 6-205
 - 6.88.4 Function Qualifiers 6-205

- 6.89 OA\$INI_SET_DEFAULT_DIR 6-206
 - 6.89.1 Syntax 6-206
 - 6.89.2 Description 6-206
 - 6.89.3 Function Parameters 6-206
 - 6.89.4 Function Qualifiers 6-206
- 6.90 OA\$INI_SUBPROCESS 6-207
 - 6.90.1 Syntax 6-207
 - 6.90.2 Description 6-207
 - 6.90.3 Function Parameters 6-207
 - 6.90.4 Function Qualifiers 6-207
 - 6.90.5 Examples 6-208
- 6.91 OA\$KEY_COMMAND 6-209
 - 6.91.1 Syntax 6-209
 - 6.91.2 Description 6-209
 - 6.91.3 Function Parameters 6-209
 - 6.91.4 Special Features and Restrictions 6-209
- 6.92 OA\$LIST_xxx 6-210
 - 6.92.1 Syntax 6-210
 - 6.92.2 Description 6-210
 - 6.92.3 Function Parameters 6-210
 - 6.92.4 Function Qualifiers 6-210
 - 6.92.5 Examples 6-211
 - 6.92.6 Special Features and Restrictions 6-211
- 6.93 OA\$LOG_DISPLAY 6-212
 - 6.93.1 Syntax 6-212
 - 6.93.2 Description 6-212
 - 6.93.3 Function Parameters 6-212
 - 6.93.4 Function Qualifiers 6-212
 - 6.93.5 Special Features and Restrictions 6-213
- 6.94 OA\$LOG_TO_FILE 6-214
 - 6.94.1 Syntax 6-214
 - 6.94.2 Description 6-214
 - 6.94.3 Function Parameters 6-214
 - 6.94.4 Function Qualifiers 6-215
 - 6.94.5 Examples 6-215
- 6.95 OA\$MENU_LEVEL_PUSH 6-216
 - 6.95.1 Syntax 6-216
 - 6.95.2 Description 6-216
 - 6.95.3 Function Parameters 6-217
 - 6.95.4 Function Qualifiers 6-217

- 6.96 OA\$MENU_SHOW_STACK 6-218
 - 6.96.1 Syntax 6-218
 - 6.96.2 Description 6-218
 - 6.96.3 Function Parameters 6-219
 - 6.96.4 Function Qualifiers 6-219
- 6.97 OA\$MRG_TTOUT 6-220
 - 6.97.1 Syntax 6-220
 - 6.97.2 Description 6-220
 - 6.97.3 Function Parameters 6-220
 - 6.97.4 Examples 6-221
- 6.98 OA\$MESSG_PURGE 6-222
 - 6.98.1 Syntax 6-222
 - 6.98.2 Description 6-222
 - 6.98.3 Function Parameters 6-222
 - 6.98.4 Examples 6-222
- 6.99 OA\$MESSG_TRAP 6-223
 - 6.99.1 Syntax 6-223
 - 6.99.2 Description 6-223
 - 6.99.3 Function Parameters 6-223
 - 6.99.4 Function Qualifiers 6-223
- 6.100 OA\$SCL_nnn 6-224
 - 6.100.1 Syntax 6-224
 - 6.100.2 Description 6-224
 - 6.100.3 Scroll Functions 6-224
 - 6.100.4 File Conversion for /SCROLL 6-227
 - 6.100.5 Special Features and Restrictions 6-227
- 6.101 OA\$SCP_STACK_DUMP 6-228
 - 6.101.1 Syntax 6-228
 - 6.101.2 Description 6-228
 - 6.101.3 Function Parameters 6-228
 - 6.101.4 Function Qualifiers 6-228
- 6.102 OA\$STAT_GET_STATISTICS 6-229
 - 6.102.1 Syntax 6-229
 - 6.102.2 Description 6-229
 - 6.102.3 Function Parameters 6-230
 - 6.102.4 Function Qualifiers 6-230
 - 6.102.5 Special Features and Restrictions 6-230
- 6.103 OA\$SUB_CLOSE 6-231
 - 6.103.1 Syntax 6-231
 - 6.103.2 Description 6-231
 - 6.103.3 Function Parameters 6-231
 - 6.103.4 Function Qualifiers 6-231

- 6.104 OA\$SUB__OPEN 6-232
 - 6.104.1 Syntax 6-232
 - 6.104.2 Description 6-232
 - 6.104.3 Function Parameters 6-232
 - 6.104.4 Function Qualifiers 6-232
- 6.105 OA\$SYM__CLOSE 6-233
 - 6.105.1 Syntax 6-233
 - 6.105.2 Description 6-233
 - 6.105.3 Function Parameters 6-233
 - 6.105.4 Function Qualifiers 6-233
- 6.106 OA\$SYM__GET__SYMBOL 6-234
 - 6.106.1 Syntax 6-234
 - 6.106.2 Description 6-234
 - 6.106.3 Function Parameters 6-234
 - 6.106.4 Function Qualifiers 6-235
 - 6.106.5 Examples 6-235
- 6.107 OA\$SYM__MAKE__SYMBOL 6-236
 - 6.107.1 Syntax 6-236
 - 6.107.2 Description 6-236
 - 6.107.3 Function Parameters 6-236
 - 6.107.4 Function Qualifiers 6-236
 - 6.107.5 Examples 6-237
- 6.108 OA\$SYM__OPEN 6-238
 - 6.108.1 Syntax 6-238
 - 6.108.2 Description 6-238
 - 6.108.3 Function Parameters 6-238
 - 6.108.4 Function Qualifiers 6-238
 - 6.108.5 Special Features and Restrictions 6-239
- 6.109 OA\$TRA__OUT 6-240
 - 6.109.1 Syntax 6-240
 - 6.109.2 Description 6-240
 - 6.109.3 Function Parameters 6-240
 - 6.109.4 Function Qualifiers 6-240
- 6.110 OA\$TRA__SET 6-241
 - 6.110.1 Syntax 6-241
 - 6.110.2 Description 6-241
 - 6.110.3 Function Parameters 6-241
 - 6.110.4 Function Qualifiers 6-242
 - 6.110.5 Examples 6-242
 - 6.110.6 Special Features and Restrictions 6-243

- 6.111 OA\$TRM_BRDCST_INIT 6-244
 - 6.111.1 Syntax 6-244
 - 6.111.2 Description 6-244
 - 6.111.3 Function Parameters 6-244
 - 6.111.4 Function Qualifiers 6-244
 - 6.111.5 Examples 6-244
- 6.111A OA\$TRM_BRDCST_STOP 6-244.2
 - 6.111A.1 Syntax 6-244.2
 - 6.111A.2 Description 6-244.2
 - 6.111A.3 Function Parameters 6-244.2
 - 6.111A.4 Function Qualifiers 6-244.2
 - 6.111A.5 Examples 6-244.3
- 6.112 OA\$TRM_INIT 6-245
 - 6.112.1 Syntax 6-245
 - 6.112.2 Description 6-245
 - 6.112.3 Function Parameters 6-245
 - 6.112.4 Function Qualifiers 6-245
- 6.113 OA\$TRM_INQUIRE 6-246
 - 6.113.1 Syntax 6-246
 - 6.113.2 Description 6-246
 - 6.113.3 Function Parameters 6-246
 - 6.113.4 Function Qualifiers 6-246
- 6.114 OA\$TRM_PORT_LINE 6-247
 - 6.114.1 Syntax 6-247
 - 6.114.2 Description 6-247
 - 6.114.3 Function Parameters 6-247
 - 6.114.4 Function Qualifiers 6-247
- 6.115 OA\$TRM_PORT_LOG_ON 6-248
 - 6.115.1 Syntax 6-248
 - 6.115.2 Description 6-248
 - 6.115.3 Function Parameters 6-248
 - 6.115.4 Function Qualifiers 6-248
- 6.116 OA\$TRM_PORT_OFF 6-249
 - 6.116.1 Syntax 6-249
 - 6.116.2 Description 6-249
 - 6.116.3 Function Parameters 6-249
 - 6.116.4 Function Qualifiers 6-249
- 6.117 OA\$TRM_PORT_ON 6-250
 - 6.117.1 Syntax 6-250
 - 6.117.2 Description 6-250
 - 6.117.3 Function Parameters 6-250
 - 6.117.4 Function Qualifiers 6-250
- 6.118 OA\$TRM_PORT_PRINT 6-251
 - 6.118.1 Syntax 6-251
 - 6.118.2 Description 6-251

6.118.3	Function Parameters	6-251
6.118.4	Function Qualifiers	6-251
6.119	OA\$TXL_CLOSE	6-252
6.119.1	Syntax	6-252
6.119.2	Description	6-252
6.119.3	Function Parameters	6-252
6.119.4	Function Qualifiers	6-252
6.119.5	Special Features and Restrictions	6-252
6.120	OA\$TXL_COMPILE	6-253
6.120.1	Syntax	6-253
6.120.2	Description	6-253
6.120.3	Function Parameters	6-253
6.120.4	Function Qualifiers	6-253
6.121	OA\$TXL_DIRECTORY	6-254
6.121.1	Syntax	6-254
6.121.2	Description	6-254
6.121.3	Function Parameters	6-254
6.121.4	Function Qualifiers	6-254
6.122	OA\$TXL_INSTALL	6-255
6.122.1	Syntax	6-255
6.122.2	Description	6-255
6.122.3	Function Parameters	6-255
6.122.4	Function Qualifiers	6-255
6.122.5	Special Features and Restrictions	6-255
6.123	OA\$TXL_LIST	6-256
6.123.1	Syntax	6-256
6.123.2	Description	6-256
6.123.3	Function Parameters	6-256
6.123.4	Function Qualifiers	6-256
6.123.5	Examples	6-256
6.124	OA\$TXL_OPEN	6-257
6.124.1	Syntax	6-257
6.124.2	Description	6-257
6.124.3	Function Parameters	6-257
6.124.4	Function Qualifiers	6-257
6.124.5	Special Features and Restrictions	6-258
6.125	OA\$TXL_REMOVE	6-259
6.125.1	Syntax	6-259
6.125.2	Description	6-259
6.125.3	Function Parameters	6-259
6.125.4	Function Qualifiers	6-259
6.125.5	Special Features and Restrictions	6-259

- 6.126 OA\$VAL__SET__VALID 6-260
 - 6.126.1 Syntax 6-260
 - 6.126.2 Description 6-260
 - 6.126.3 Function Parameters 6-260
 - 6.126.4 Function Qualifiers 6-260
 - 6.126.5 Examples 6-261
- 6.127 PARSE__USER 6-262
 - 6.127.1 Syntax 6-262
 - 6.127.2 Description 6-262
 - 6.127.3 Function Parameters 6-262
 - 6.127.4 Function Qualifiers 6-263
 - 6.127.5 Examples 6-263
- 6.128 PORT__ON, PORT__OFF 6-264
 - 6.128.1 Syntax 6-264
 - 6.128.2 Description 6-264
 - 6.128.3 Function Parameters 6-264
 - 6.128.4 Function Qualifiers 6-264
 - 6.128.5 Examples 6-264
 - 6.128.6 Special Features and Restrictions 6-264
- 6.129 PRINT__SCREEN 6-265
 - 6.129.1 Syntax 6-265
 - 6.129.2 Description 6-265
 - 6.129.3 Function Parameters 6-266
 - 6.129.4 Function Qualifiers 6-266
 - 6.129.5 Examples 6-266
- 6.130 PROMPT 6-267
 - 6.130.1 Syntax 6-267
 - 6.130.2 Description 6-267
 - 6.130.3 Function Parameters 6-268
 - 6.130.4 Function Qualifiers 6-268
 - 6.130.5 Examples 6-268
 - 6.130.6 Special Features and Restrictions 6-269
- 6.131 PROMPT__ID 6-271
 - 6.131.1 Syntax 6-271
 - 6.131.2 Description 6-271
 - 6.131.3 Function Parameters 6-272
 - 6.131.4 Function Qualifiers 6-272
- 6.132 PURGE__FILE 6-273
 - 6.132.1 Syntax 6-273
 - 6.132.2 Description 6-273
 - 6.132.3 Function Parameters 6-273
 - 6.132.4 Function Qualifiers 6-274
 - 6.132.5 Examples 6-274
 - 6.132.6 Special Features and Restrictions 6-275

6.133	PUT_FIELD	6-277
6.133.1	Syntax	6-277
6.133.2	Description	6-277
6.133.3	Function Parameters	6-277
6.133.4	Function Qualifiers	6-278
6.133.5	Examples	6-278
6.133.6	Special Features and Restrictions	6-278
6.134	QUEUE_BATCH	6-279
6.134.1	Syntax	6-279
6.134.2	Description	6-279
6.134.3	Function Parameters	6-279
6.134.4	Function Qualifiers	6-279
6.134.5	Examples	6-288
6.134.6	Special Features and Restrictions	6-282
6.135	QUEUE_PRINT	6-284
6.135.1	Syntax	6-284
6.135.2	Description	6-284
6.135.3	Function Parameters	6-284
6.135.4	Function Qualifiers	6-284
6.135.5	Examples	6-287
6.135.6	Special Features and Restrictions	6-288
6.136	RENAME	6-289
6.136.1	Syntax	6-289
6.136.2	Description	6-289
6.136.3	Function Parameters	6-289
6.136.4	Function Qualifiers	6-289
6.136.5	Special Features and Restrictions	6-290
6.137	REPEAT	6-291
6.137.1	Syntax	6-291
6.137.2	Description	6-291
6.137.3	Function Parameters	6-291
6.137.4	Function Qualifiers	6-291
6.137.5	Examples	6-291
6.137.6	Special Features and Restrictions	6-292
6.138	SAVE	6-293
6.138.1	Syntax	6-293
6.138.2	Description	6-293
6.138.3	Function Parameters	6-293
6.138.4	Function Qualifiers	6-294
6.138.5	Examples	6-294
6.138.6	Special Features and Restrictions	6-294

6.139	SCRIPT	6-296
6.139.1	Syntax	6-296
6.139.2	Description	6-296
6.139.3	Function Parameters	6-296
6.139.4	Function Qualifiers	6-297
6.139.5	Examples	6-297
6.139.6	Special Features and Restrictions	6-297
6.140	SET_DEFAULT	6-298
6.140.1	Syntax	6-298
6.140.2	Description	6-298
6.140.3	Function Parameters	6-298
6.140.4	Function Qualifiers	6-298
6.140.5	Examples	6-299
6.140.6	Special Features and Restrictions	6-299
6.141	SET_MENU	6-300
6.141.1	Syntax	6-300
6.141.2	Description	6-300
6.141.3	Function Parameters	6-300
6.141.4	Function Qualifiers	6-300
6.141.5	Examples	6-301
6.141.6	Special Features and Restrictions	6-301
6.142	SHOW_FORM	6-302
6.142.1	Syntax	6-302
6.142.2	Description	6-302
6.142.3	Function Parameters	6-302
6.142.4	Function Qualifiers	6-302
6.143	SORT_LIST	6-303
6.143.1	Syntax	6-303
6.143.2	Description	6-303
6.143.3	Function Parameters	6-303
6.143.4	Function Qualifiers	6-305
6.143.5	Examples	6-305
6.143.6	Special Features and Restrictions	6-305
6.144	SPAWN	6-306
6.144.1	Syntax	6-306
6.144.2	Description	6-306
6.144.3	Function Parameters	6-306
6.144.4	Function Qualifiers	6-307
6.144.5	Special Features and Restrictions	6-307

6.145	STATISTICS	6-308
6.145.1	Syntax	6-308
6.145.2	Description	6-308
6.145.3	Function Parameters	6-308
6.145.4	Function Qualifiers	6-309
6.145.5	Examples	6-309
6.146	TEST_SPEC	6-310
6.146.1	Syntax	6-310
6.146.2	Description	6-310
6.146.3	Function Parameters	6-310
6.146.4	Function Qualifiers	6-310
6.146.5	Examples	6-311
6.146.6	Special Features and Restrictions	6-311
6.147	TIME	6-312
6.147.1	Syntax	6-312
6.147.2	Description	6-312
6.147.3	Function Parameters	6-312
6.147.4	Function Qualifiers	6-312
6.147.5	Examples	6-313
6.148	UNWIND	6-314
6.148.1	Syntax	6-314
6.148.2	Description	6-314
6.148.3	Function Parameters	6-314
6.148.4	Function Qualifiers	6-314
6.148.5	Examples	6-315
6.148.6	Special Features and Restrictions	6-315
6.149	VERSION	6-316
6.149.1	Syntax	6-316
6.149.2	Description	6-316
6.149.3	Function Parameters	6-316
6.149.4	Function Qualifiers	6-316
6.149.5	Examples	6-317
6.150	WAIT	6-318
6.150.1	Syntax	6-318
6.150.2	Description	6-318
6.150.3	Function Parameters	6-318
6.150.4	Function Qualifiers	6-319
6.150.5	Examples	6-319
6.150.6	Special Features and Restrictions	6-320

- 6.151 WPSPLUS\$CBI__CLOSE__WINDOW 6-321
 - 6.151.1 Syntax 6-321
 - 6.151.2 Description 6-321
 - 6.151.3 Function Parameters 6-321
 - 6.151.4 Function Qualifiers 6-321
 - 6.151.5 Examples 6-321
 - 6.151.6 Special Features and Restrictions 6-321
- 6.152 WPSPLUS\$CBI__CLOSE__WINDOW2 6-322
 - 6.152.1 Syntax 6-322
 - 6.152.2 Description 6-322
 - 6.152.3 Function Parameters 6-322
 - 6.152.4 Function Qualifiers 6-322
 - 6.152.5 Examples 6-322
 - 6.152.6 Special Features and Restrictions 6-322
- 6.153 WPSPLUS\$CBI__OPEN__WINDOW 6-323
 - 6.153.1 Syntax 6-323
 - 6.153.2 Description 6-323
 - 6.153.3 Function Parameters 6-323
 - 6.153.4 Function Qualifiers 6-323
 - 6.153.5 Examples 6-323
 - 6.153.6 Special Features and Restrictions 6-324
- 6.154 WPSPLUS\$CBI__OPEN__WINDOW2 6-325
 - 6.154.1 Syntax 6-325
 - 6.154.2 Description 6-325
 - 6.154.3 Function Parameters 6-325
 - 6.154.4 Function Qualifiers 6-325
 - 6.154.5 Examples 6-326
 - 6.154.6 Special Features and Restrictions 6-326
- 6.155 WPSPLUS__SPELL 6-327
 - 6.155.1 Syntax 6-327
 - 6.155.2 Description 6-327
 - 6.155.3 Function Parameters 6-327
 - 6.155.4 Function Qualifiers 6-327
 - 6.155.5 Examples 6-327
 - 6.155.6 Special Features and Restrictions 6-327
- 6.156 WRITE 6-328
 - 6.156.1 Syntax 6-328
 - 6.156.2 Description 6-328
 - 6.156.3 Function Parameters 6-328
 - 6.156.4 Function Qualifiers 6-329
 - 6.156.5 Examples 6-329
 - 6.156.6 Special Features and Restrictions 6-329

6.157	YESNO_PROMPT	6-330
6.157.1	Syntax	6-330
6.157.2	Description	6-330
6.157.3	Function Parameters	6-330
6.157.4	Function Qualifiers	6-331
6.157.5	Examples	6-331
6.157.6	Special Features and Restrictions	6-332
6.158	YESNO_PROMPT_ID	6-333
6.158.1	Syntax	6-333
6.158.2	Description	6-333
6.158.3	Function Parameters	6-333
6.158.4	Function Qualifiers	6-334
6.158.5	Special Features and Restrictions	6-334

Volume 3: Applications

Preface

7 The Subprocess

- 7.1 Introduction 7-1
- 7.2 Initializing the Subprocess 7-1
- 7.3 Communicating with the Subprocess Through Mailboxes 7-2
 - 7.3.1 SYS\$INPUT 7-4
 - 7.3.2 OAMAILBOX and DCLMAILBOX 7-4
- 7.4 WRITE OAMAILBOX Syntax 7-6
- 7.5 Defining and Using Symbols 7-8
 - 7.5.1 Creating DCL Symbols with /COPY and /PUT_SAVE 7-9
 - 7.5.2 Creating DCL Symbols with Functions 7-10
- 7.6 Common DCL Symbols 7-12
 - 7.6.1 Passing Command-Procedure Parameters to the Subprocess from ALL-IN-1 7-16
- 7.7 ALL-IN-1 Logicals 7-16
- 7.8 Running Command Procedures and Programs 7-19
 - 7.8.1 Running Command Procedures 7-19
 - 7.8.2 Obtaining User Input in a Program or Command Procedure 7-20

8 ALL-IN-1 Symbols

- 8.1 Introduction 8-1
- 8.2 Basic Symbol Types 8-2
 - 8.2.1 Quoted String 8-2
 - 8.2.2 Field Name 8-2
 - 8.2.3 Permanent Symbol Table 8-2
 - 8.2.4 Special Symbols 8-3
 - 8.2.5 Local Symbol Table 8-4
 - 8.2.6 Data Set References 8-4

- 8.3 Using Data Set References 8-5
 - 8.3.1 Combining and Nesting DSRs 8-6
 - 8.3.2 Specifying Alternate Keys with DSRs 8-6
 - 8.3.3 Special Field Names 8-7
- 8.4 Symbol Substrings 8-10
- 8.5 Symbol Lists 8-11
- 8.6 Iterative Symbol Substitution 8-12
- 8.7 Specifying Date/Time Values 8-13
 - 8.7.1 Creating and Updating Symbols and Data Sets 8-14
 - 8.7.2 Getting Values from Symbols 8-15
 - 8.7.3 Values Stored in Multiple Symbols 8-15
- 8.8 Special Symbols 8-15
- 8.9 Permanent Symbols 8-42

9 The File Cabinet

- 9.1 File Cabinet Concepts 9-2
- 9.2 File Cabinet Organization 9-4
 - 9.2.1 OAUSER:DOCDB.DAT, the Personal File Cabinet Index 9-4
- 9.3 Document Attributes 9-8
 - 9.3.1 The Document Attributes Files 9-9
 - 9.3.2 Document Attributes 9-9
 - 9.3.3 The System Pending File 9-12
- 9.4 Using the File Cabinet 9-13
 - 9.4.1 Summary of CABINET Subfunctions 9-13
 - 9.4.2 Accessing File Cabinet Data 9-15
 - 9.4.3 The Shared Area 9-19
 - 9.4.4 Shared Libraries 9-19
- 9.5 CABINET Subfunctions 9-20

10 Text Processing

- 10.1 Overview 10-1
- 10.2 Text Data Sets 10-2
- 10.3 Creating Files with the EDIT function 10-3
- 10.4 Features of the Default Editor 10-4
 - 10.4.1 Editor Modes 10-4
 - 10.4.2 Communicating with the Rest of ALL-IN-1 Through the Editor 10-10
 - 10.4.3 Moving Within a Document 10-13
 - 10.4.4 Moving Text Within a Document 10-14
 - 10.4.5 Including Outside Text into a Document 10-16
 - 10.4.6 Creating New Files While in the Editor 10-17
 - 10.4.7 Customizing the Default Editor 10-18

10.5 Features of the WPS-PLUS Editor	10-21
10.6 List Processing	10-21
10.7 Formatting and Outputting Text	10-28

11 Electronic Messaging

11.1 Introduction	11-1
11.2 Electronic Messaging System Concepts	11-1
11.2.1 Current Message	11-1
11.2.2 Message Header and Text	11-2
11.2.3 Mail Interfaces	11-2
11.2.4 Mail Folders	11-2
11.2.5 Nicknames and Distribution Lists	11-3
11.2.6 Mail Delivery Mechanism	11-3
11.2.7 Alternative Message Transport Systems (MTS)	11-5
11.3 Special Symbols	11-9
11.4 Customizing the Message Header and Show-Message Displays	11-9
11.5 MAIL Subfunctions	11-10
11.5.1 ANSWER	11-11
11.5.2 ATTACH	11-13
11.5.3 AUTO_FORWARD	11-13
11.5.4 AUTO_REPLY	11-14
11.5.5 CANCEL	11-14
11.5.6 CANCEL_AUTO_REPLY	11-15
11.5.7 CC	11-15
11.5.8 CLOSE_MESSAGE	11-16
11.5.9 CREATE	11-16.1
11.5.10 DEFAULT_MESSAGE	11-19
11.5.11 DEFER	11-20
11.5.12 DELETE	11-20
11.5.13 DELIVERY_RECEIPT	11-21
11.5.14 DETACH	11-22
11.5.15 DIST_COPY	11-22
11.5.16 DIST_CREATE	11-23
11.5.17 DIST_DELETE	11-23
11.5.18 DIST_EDIT	11-24
11.5.19 DIST_LIST	11-24
11.5.20 EDIT	11-25
11.5.21 EXCHANGE_CURMES	11-26
11.5.22 EXPAND_DIST_LIST	11-26
11.5.23 FILE_ATTACHMENT	11-27

11.5.24	FILE_BODY	11-28
11.5.25	FILE_MESSAGE	11-29
11.5.26	FORWARD	11-30
11.5.27	FORWARDABLE	11-32
11.5.28	FUNCTION	11-32
11.5.29	GET	11-33
11.5.30	INDEX	11-33
11.5.31	MAIL_DOCUMENT	11-34
11.5.32	NEXT	11-36
11.5.33	NO_DELIVERY_RECEIPT	11-37
11.5.34	NON_FORWARDABLE	11-37
11.5.35	ORIGINAL	11-37
11.5.36	OUTBOX	11-38
11.5.37	POP_CURMES	11-38
11.5.38	PRINT	11-39
11.5.39	PRIORITY	11-40
11.5.40	PUSH_CURMES	11-41
11.5.41	READ	11-42
11.5.42	READ_RECEIPT	11-43
11.5.43	SEND	11-44
11.5.44	SHOW	11-46
11.5.45	STATUS	11-46
11.5.46	SUBJECT	11-47
11.5.47	TEXT	11-48
11.5.48	TO	11-48
11.5.49	VAXMAIL_CHECK	11-49
11.6	Electronic Messaging Data Set References	11-50

12 External Communications

12.1	Overview	12-1
12.2	Multinode ALL-IN-1 Applications	12-1
12.3	The Communications Control Subsystem	12-2
12.3.1	Communications Control Concepts	12-2
12.3.2	CXP Symbols	12-2
12.3.3	CXP Subfunctions	12-5
12.3.4	CXP Subfunctions that Differ from Control Document Commands	12-9
12.3.5	Using Communication Control Commands	12-11
12.3.6	Example of Customizing Communications Control	12-12

13 Time Management

- 13.1 Introduction 13-1**
- 13.2 Organization 13-1**
 - 13.2.1 Time Management Activities 13-2**
 - 13.2.2 Subfunctions and the Current CALENDAR Date/Time 13-2**
 - 13.2.3 CALENDAR Files 13-3**
- 13.3 CALENDAR Subfunctions 13-3**
 - 13.3.1 CALENDAR ADD__ATTENDEE 13-3**
 - 13.3.2 CALENDAR CANCEL APPOINTMENT 13-4**
 - 13.3.3 CALENDAR CANCEL MEETING 13-4**
 - 13.3.4 CALENDAR CLEAR ATTENDEES 13-5**
 - 13.3.5 CALENDAR CONVERT APPOINTMENTS 13-5**
 - 13.3.6 CALENDAR CONVERT ACTION_ITEMS 13-5**
 - 13.3.7 CALENDAR CONVERT MEETINGS 13-6**
 - 13.3.8 CALENDAR DISPLAY ANSWER 13-6**
 - 13.3.9 CALENDAR DISPLAY APPOINTMENT 13-7**
 - 13.3.10 CALENDAR DISPLAY GRAPH 13-8**
 - 13.3.11 CALENDAR DISPLAY MEETING 13-9**
 - 13.3.12 CALENDAR DISPLAY MONTH 13-9**
 - 13.3.13 CALENDAR DISPLAY NEXT APPOINTMENT 13-10**
 - 13.3.14 CALENDAR DISPLAY NEXT MEETING 13-11**
 - 13.3.15 CALENDAR DISPLAY NEXT REPLY 13-11**
 - 13.3.16 CALENDAR DISPLAY REPLY 13-12**
 - 13.3.17 CALENDAR INITIALIZE FILES 13-13**
 - 13.3.18 CALENDAR INITIALIZE MONTH 13-13**
 - 13.3.19 CALENDAR MAIL__TO__ATTENDEES 13-14**
 - 13.3.20 CALENDAR MAKE__ATTENDEE__SYMBOLS 13-14**
 - 13.3.21 CALENDAR MAKE__MEETING__SYMBOLS 13-15**
 - 13.3.22 CALENDAR PRINT ALL 13-16**
 - 13.3.23 CALENDAR PRINT CONFLICTS 13-16**
 - 13.3.24 CALENDAR PRINT CUSTOM 13-17**
 - 13.3.24A CALENDAR PRINT DAY_REMIND 13-17**
 - 13.3.25 CALENDAR PRINT DAY_SCHEDULE 13-18**
 - 13.3.26 CALENDAR PRINT MONTH 13-18**
 - 13.3.26A CALENDAR PRINT TODO 13-19**
 - 13.3.27 CALENDAR PRINT TWOCAL 13-19**
 - 13.3.28 CALENDAR PRINT WEEK_REMINDERS 13-20**
 - 13.3.29 CALENDAR PRINT WEEK_SCHEDULE 13-20**
 - 13.3.30 CALENDAR PURGE 13-20.1**

- 13.3.31 CALENDAR REMOVE_ATTENDEES 13-20.1
- 13.3.32 CALENDAR REPLY_TO_MEETING 13-21
- 13.3.33 CALENDAR RESCHEDULE
APPOINTMENT 13-21
- 13.3.34 CALENDAR RESCHEDULE MEETING 13-22
- 13.3.35 CALENDAR SCAN 13-22
- 13.3.36 CALENDAR SCHEDULE APPOINTMENT 13-23
- 13.3.37 CALENDAR SCHEDULE MEETING 13-23
- 13.3.38 CALENDAR SET ALTERNATE_USER 13-24
- 13.3.39 CALENDAR SET DATE 13-25
- 13.3.40 CALENDAR SET NETWORK OFF 13-26
- 13.3.41 CALENDAR SET NETWORK ON 13-26
- 13.3.42 CALENDAR VERSION 13-27
- 13.4 Calendar File Layouts 13-27
 - 13.4.1 User File Layout 13-27
 - 13.4.2 Meeting File Layout 13-29
 - 13.4.3 Attendee File Layout 13-30
- 13.5 CALENDAR Data Sets 13-30

14 The Help and Message Facilities

- 14.1 Introduction 14-1
- 14.2 Help Facility 14-1
 - 14.2.1 The HELP Function 14-1
 - 14.2.2 The Help Library 14-6
 - 14.2.3 Maintaining, Updating, and Switching Between
Help Libraries 14-18
 - 14.2.4 Help at a Prompt 14-22
- 14.3 Message Facility 14-22
 - 14.3.1 The Message File 14-23
 - 14.3.2 Updating the Message File 14-23
 - 14.3.3 Message Severity and the Message Buffer 14-24
 - 14.3.4 Help on Error Messages 14-26

15 The Script Facility

- 15.1 Overview 15-1
- 15.2 Scripts 15-1
 - 15.2.1 Script Modes 15-2
 - 15.2.2 Interactive Directives and the
Pseudo-Input Buffer 15-3
 - 15.2.3 Exiting from Scripts 15-4
 - 15.2.4 ALL-IN-1 TXLs 15-5
 - 15.2.5 Script-File Format 15-7

- 15.3 Script Directives 15-9
- 15.4 Script Directives as Functions 15-31
- 15.5 Sample Script 15-32
- 15.6 User Defined Processes 15-33
- 15.7 Running ALL-IN-1 Indirectly or in Batch Mode 15-33
- 15.8 Special Features and Restrictions 15-33
 - 15.8.1 Creating a Script with a Script 15-33
 - 15.8.2 UDPs for the Desk Calculator 15-34

16 Application Development Tools and Design Guidelines

- 16.1 Introduction 16-1
 - 16.1.1 Where to Start 16-2
- 16.2 The Forms Development Subsystem 16-2
 - 16.2.1 Using Form Libraries 16-9
 - 16.2.2 Testing Newly Created Forms 16-11
- 16.3 The Programming Functions Subsystem 16-12
- 16.4 Other Development Tools 16-14
 - 16.4.1 Debugging Features 16-15
 - 16.4.2 Using the Subprocess Interactively 16-15
 - 16.4.3 Using the Editor 16-16
 - 16.4.4 Using the GET Function 16-17
 - 16.4.5 Using View Forms 16-18
- 16.5 Application Design Guidelines 16-18
 - 16.5.1 Standard Applications 16-19
 - 16.5.2 Standalone Applications 16-20
 - 16.5.3 User-Defined Functions 16-21
- 16.6 Performance Considerations 16-22
- 16.7 Documenting a Customized System 16-23
 - 16.7.1 User Documentation 16-23
 - 16.7.2 Help Documentation 16-24

A Exercises

- A.1 Introduction A-1
- A.2 Exercise 1 — Using Standard ALL-IN-1 Subsystems A-2
 - A.2.1 Objective A-2
 - A.2.2 Preparation A-2
 - A.2.3 Exercise A-2
 - A.2.4 Summary A-3
 - A.2.5 References A-3

A.3	Exercise 2 — Modifying the User Profile	A-3
A.3.1	Objective	A-3
A.3.2	Preparation	A-3
A.3.3	Exercise	A-3
A.3.4	Summary	A-4
A.3.5	References	A-4
A.4	Exercise 3 — Advanced User Tasks	A-4
A.4.1	Objective	A-4
A.4.2	Preparation	A-4
A.4.3	Exercise	A-4
A.4.4	Summary	A-5
A.4.5	References	A-5
A.5	Exercise 4 — Modifying the User Profile (Programmer Level)	A-5
A.5.1	Objective	A-5
A.5.2	Preparation	A-6
A.5.3	Exercise	A-6
A.5.4	Summary	A-6
A.5.5	References	A-7
A.6	Exercise 5 — Accessing the Subprocess	A-7
A.6.1	Objective	A-7
A.6.2	Preparation	A-7
A.6.3	Exercise	A-7
A.6.4	Summary	A-8
A.6.5	References	A-8
A.7	Exercise 6 — Using Functions Interactively	A-8
A.7.1	Objective	A-8
A.7.2	Preparation	A-8
A.7.3	Exercise	A-8
A.7.4	Summary	A-9
A.7.5	References	A-9
A.8	Exercise 7 — Using the Hard Copy Interface	A-9
A.8.1	Objective	A-9
A.8.2	Preparation	A-9
A.8.3	Exercise	A-10
A.8.4	Summary	A-11
A.8.5	References	A-11
A.9	Exercise 8 — Using Development Tools	A-11
A.9.1	Objective	A-11
A.9.2	Preparation	A-11
A.9.3	Exercise	A-11
A.9.4	Summary	A-12
A.9.5	References	A-12

- A.10 Exercise 9 — Modifying a Menu A-12
 - A.10.1 Objective A-12
 - A.10.2 Preparation A-12
 - A.10.3 Exercises A-12
 - A.10.4 Summary A-13
 - A.10.5 References A-13
- A.11 Exercise 10 — Creating an Entry Form for an Indexed File A-13
 - A.11.1 Objective A-13
 - A.11.2 Preparation A-14
 - A.11.3 Exercise A-14
 - A.11.4 Summary A-15
 - A.11.5 References A-15
- A.12 Exercise 11 — Creating a Select Form A-15
 - A.12.1 Objective A-15
 - A.12.2 Preparation A-15
 - A.12.3 Exercise A-15
 - A.12.4 Summary A-16
 - A.12.5 References A-16
- A.13 Exercise 12 — Modifying a Script A-16
 - A.13.1 Objective A-16
 - A.13.2 Preparation A-17
 - A.13.3 Exercise A-17
 - A.13.4 Summary A-17
 - A.13.5 References A-17
 - A.13.6 Answers A-18
- A.14 Exercise 13 — Creating a Shared Template A-19
 - A.14.1 Objective A-19
 - A.14.2 Preparation A-19
 - A.14.3 Exercise A-19
 - A.14.4 Summary A-20
 - A.14.5 References A-20
- A.15 Exercise 14 — Using DATATRIEVE A-20
 - A.15.1 Objective A-20
 - A.15.2 Preparation A-21
 - A.15.3 Exercise A-21
 - A.15.4 Summary A-22
 - A.15.5 References A-22

B ALL-IN-1 Hard Copy

- B.1 ALL-IN-1 Hard Copy B-1
 - B.1.1 Intended Use B-1
 - B.1.2 Terminal Support B-2

B.1.3	Function Keys	B-3
B.1.4	Appearance on the Screen	B-3
B.1.5	/HARD Qualifiers	B-4
B.1.6	Function-Key Entry	B-5
B.1.7	Field-Full Condition	B-5

C Data Sets

C.1	Overview	C-1
C.2	Special Data Sets	C-2
C.2.1	OA\$DATE	C-2
C.2.2	OA\$DIR	C-3
C.2.3	OA\$DIR_PARSE	C-6
C.2.4	OA\$EDITOR	C-7
C.2.5	OA\$MAIDES	C-7
C.2.6	OA\$MAIL_ADDRESS	C-8
C.2.7	OA\$MAILPRIO	C-8
C.2.8	OA\$TABLE	C-8
C.2.9	OA\$YESNO	C-9
C.2.10	OA\$YN	C-9
C.3	Calendar Validation Data Sets	C-9
C.3.1	CAL\$ATTENDEE	C-9
C.3.2	CAL\$CALENDAR	C-10
C.3.3	CAL\$END_TIME_CHECK:key	C-10
C.3.4	CAL\$MEETING	C-10
C.3.5	CAL\$MTG_TIME_CHECK	C-10
C.3.6	CAL\$SET_DATE	C-10
C.3.7	CAL\$SET_OWNER	C-11
C.3.8	CAL\$SET_SECOND	C-11
C.3.9	CAL\$TIME_CHECK	C-11
C.4	Calendar Scrolling Data Sets	C-11
C.4.1	CAL\$SCROLL_BOTH	C-12
C.4.2	CAL\$SCROLL_CONFLICTS	C-12
C.4.3	CAL\$SCROLL_CUSTOM	C-12
C.4.4	CAL\$SCROLL_DAY	C-12
C.4.5	CAL\$SCROLL_TWOCAL	C-12
C.4.6	CAL\$SCROLL_WEEK_REM	C-13
C.4.7	CAL\$SCROLL_WEEK_SCH	C-13
C.5	General Scrolling Data Sets	C-13
C.5.1	SCROLL\$FUNCTION	C-13
C.6	Data Set Statistics	C-13

D Mnemonics for Script Input

Glossary

Figures

	How To Use this APR	xiv
	ALL-IN-1 Bookshelf	xvi
1-1	The VAX Information Architecture	1-2
1-2	ALL-IN-1 and the VAX Information Architecture	1-4
1-3	Form and Function	1-5
2-1	Primary ALL-IN-1 Flow-Control Framework	2-4
2-2	Flow Control Path: Display Main Menu	2-6
3-1	Menu Processing Flow Control	3-30
3-2	Form WPSEARCH and assumed Named Data	3-62
5-1	A Subsystem Menu	5-2
5-2	A Select Form	5-4
5-3	A Create Form	5-5
5-4	A Secondary Menu Form	5-8
7-1	Flow Control Between the Subprocess and Main Process	7-3
8-1	Obtaining Data with a DSR	8-9
9-1	File Cabinet Organization	9-13
10-1	Key Numbers for the Default Editor	10-12
10-2	Entry Form FORMAT (Document Handling and File Processing)	10-29
16-1	Forms Development Menu	16-3
16-2	Programming Functions Menu	16-13
16-3	Level-1 HELP Module Template	16-28
16-4	Level-2 HELP Module Template	16-29
16-5	Level-3 HELP Module Template	16-29

Tables

3-1a	Form AIE: Directives	3-40
3-1b	Form AIE: Field Information	3-41
3-2	File OA\$LIB:ACTITEM.FDL	3-43
4-1	SCROLL\$FUNCTION Actions	4-37
4-2	Control-Function Calls to Internal Terminator Functions	4-67
4-3	Terminator Functions, Dispositions, and OA\$FORM__DISPOSE	4-68
4-4	OA\$FORM__TERMINATOR Values	4-69

6-1	COMPUTE Operators and Operands	6-38
6-2	DATE_CONVERT Parameters	6-49
6-3	Relational Operators	6-85
6-4	Boolean Expressions	6-86.1
6-5	Recognized Editor Styles	6-131
7-1	ALL-IN-1 Subprocess Logicals	7-18
8-1	Special Symbols	8-16
8-2	Permanent Symbols	8-42
9-1a	Document Attributes	9-10
9-1b	Extended Attributes	9-11
10-1	Recognized Editor Styles	10-3
10-2	File Processors	10-32
B-1	Supported Terminals	B-2
C-1	Data Set Statistics (OA\$DSA_CACHE_STATUS output)	C-14

Preface

Purpose of this Guide

The *ALL-IN-1 Office Menu Application Programmer's Reference* (APR) gives a detailed overview of ALL-IN-1 and presents the tools available to the applications programmer for modifying or adding new applications. The APR also provides Application Design Guidelines and Exercises.

Intended Audience

The APR is written for the applications programmer familiar with VAX/VMS and its layered products. References to other documents that provide additional information are presented in the ALL-IN-1 Bookshelf illustrated in this Preface. References are also noted, where relevant, throughout this APR.

Before creating an application, you should be able to:

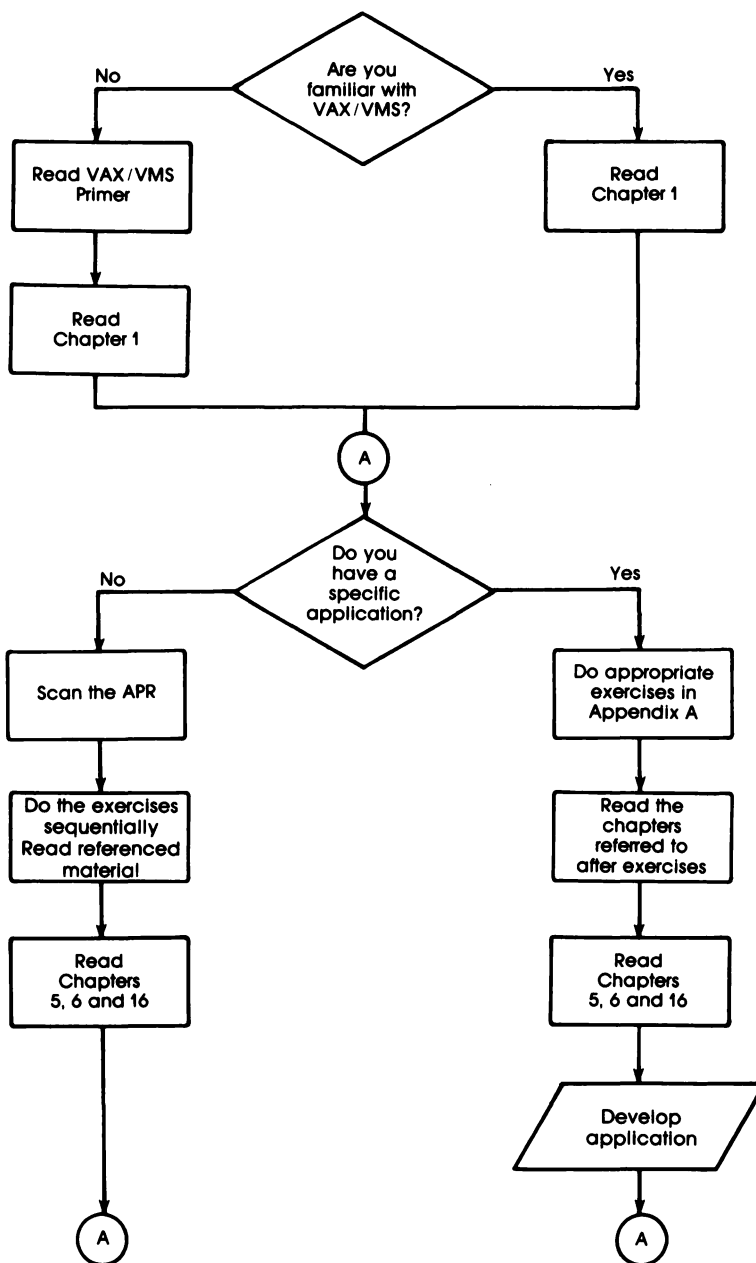
- Understand the following VAX/VMS concepts:
 - DCL
 - directory
 - file type
 - process
 - protection
 - shareable image

- library
 - logical
 - privilege
- subprocess
 - symbol
 - user identification code
- Login to a VMS system.
- Edit a file using the EDT text editor.
- Create a simple DIGITAL Command Language (DCL) command procedure.
- Create a Forms Management System (FMS) form, and understand the meaning and use of Named Data within a form.
- Create and access subdirectories.
- Create DCL symbols and assign logical names.

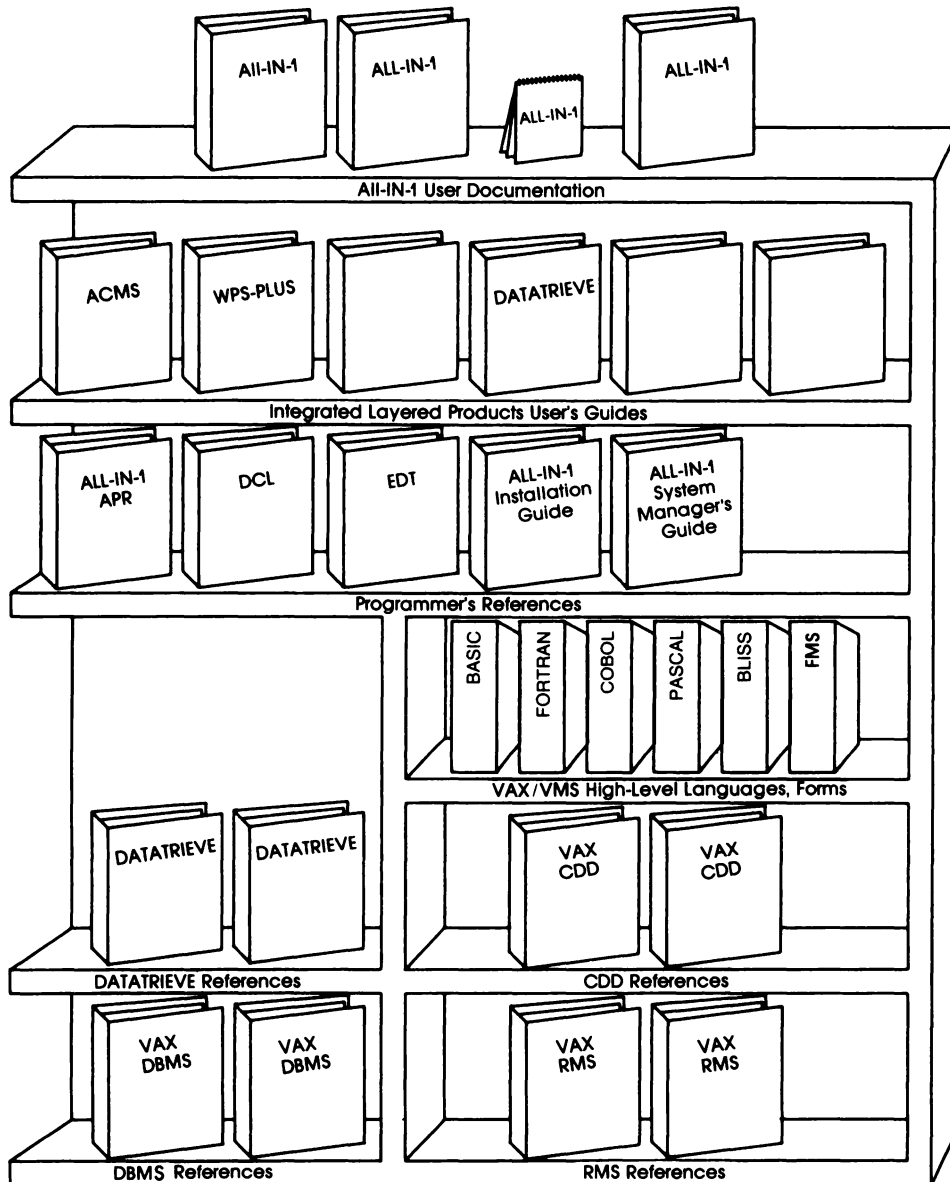
If you are unfamiliar with these concepts, refer to the flowchart on the following page and to the Glossary at the end of this guide. The flowchart may also be useful if you are familiar with VAX/VMS concepts.

How the APR is Organized

The APR has three volumes: Volume 1, Flow Control, covers the kernel of ALL-IN-1. Volume 2, ALL-IN-1 Functions, describes the functions. These are calls to ALL-IN-1 facilities to request that a specific action be performed. Volume 3, Applications, covers those closely integrated applications that use the Flow Control facilities.



MK-02000-00



This Bookshelf contains all the documents referenced throughout the APR. These documents are also listed in the Related Documents section that follows.

Related Documents

The following documents are part of the ALL-IN-1 Office Menu Documentation Set:

User Documentation

ALL-IN-1 Office Menu Getting Started Guide

ALL-IN-1 Office Menu User's Reference

Volume 1

Volume 2

ALL-IN-1 Office Menu Quick Lookup

Programmer Documentation

ALL-IN-1 Office Menu Installation Guide

ALL-IN-1 Office Menu System Manager's Guide

ALL-IN-1 Office Menu Release Notes

ALL-IN-1 Office Menu Read Me First

ALL-IN-1 Office Menu Application Programmer's Reference

Volume 1

Volume 2

Volume 3

ALL-IN-1 Office Menu Programmer's Mini-Reference

Customization Documentation

ALL-IN-1 Documenter's Kit

ALL-IN-1 Office Menu Style Guide

ALL-IN-1 Office Menu Writer's Guide

Table of Getting Started Guide and User's Reference

The following documents cover VAX/VMS concepts and VMS layered products that are integrated into or accessible from ALL-IN-1.

Application Control and Management System (ACMS)

VAX ACMS Application Management Guide

DATATRIEVE (DTR)/Database (DBMS)/Common Data Dictionary (CDD)

VAX CDD Utilities Reference Manual

Introduction to VAX DATATRIEVE

VAX DATATRIEVE User's Guide

VAX DATATRIEVE Reference Manual

VAX DATATRIEVE Guide to Using Graphics

VAX DBMS Introduction to Data Manipulation

VAX DBMS Database Administration Reference Manual

VAX DBMS Programming Reference Manual

WPS-PLUS Editor

WPS-PLUS/VMS Getting Started

WPS-PLUS/VMS Editor Functions

WPS-PLUS/VMS List and Sort Processing

WPS-PLUS/VMS Quick Lookup

EDT Editor

VAX EDT Reference Manual

Forms Management System (FMS)

VAX-11 FMS Utilities Manual

Programming Languages

VAX-11 BASIC Language Reference Manual
VAX-11 BLISS-32 Language Reference Manual
VAX-11 COBOL Language Reference Manual
VAX-11 FORTRAN Language Reference Manual
VAX-11 PASCAL Language Reference Manual
VAX-11 PL/1 Language Reference Manual

VMS Operating System

VAX/VMS Document Set
VAX/VMS Release Notes

Conventions

The following conventions are used throughout this document:

Dot Matrix	Indicates prompts and messages from the computer.
lowercase	Lowercase text in an ALL-IN-1 function call, or in a DCL command or command procedure, indicates that you substitute a valid expression in that place.
UPPERCASE	Indicates keyboard keys and Gold functions.
Enter	Indicates information the user puts into a menu or form.
Press	Indicates that the user invokes a single keyboard command (for example, press EXIT SCREEN).
Type	Indicates characters the user puts into a document, program module, or mail message from the keyboard.
Gold X	Indicates that the user presses the Gold key and then the X key to invoke a keyboard command.
CTRL/X	Indicates that the user holds down the CTRL key while pressing X to invoke a keyboard command.

[]	Indicates an optional expression in ALL-IN-1 function calls and DCL command strings.
.	Indicates that the example contains more data than is shown.
...	Indicates that the user can enter additional parameter values or information.
< >	Are used in ALL-IN-1 boiler plate forms to set off areas that contain data to be processed (for example, <OA\$TIME>).
	Indicates alternatives (for example, /INT_CHARACTER={A B D} means A, or B, or D).

1.1 Introduction

To be complete, an office automation system must provide a meaningful working environment for a spectrum of users — from clerical support people and managers to software developers and engineers. Also, since the needs of each individual and of the office as a whole varies over time, the system must be open-ended and flexible. Finally, this multi-level, open-ended system must present a common, consistent interface to all users. The ALL-IN-1 Office Menu is such a system.

ALL-IN-1 is a VAX/VMS user-mode program capable of interconnecting and integrating existing VMS layered products, third-party products, and user-designed applications. The system's capabilities derive both from built-in functionality and from the tools of the VAX Information Architecture, which is represented in Figure 1-1.

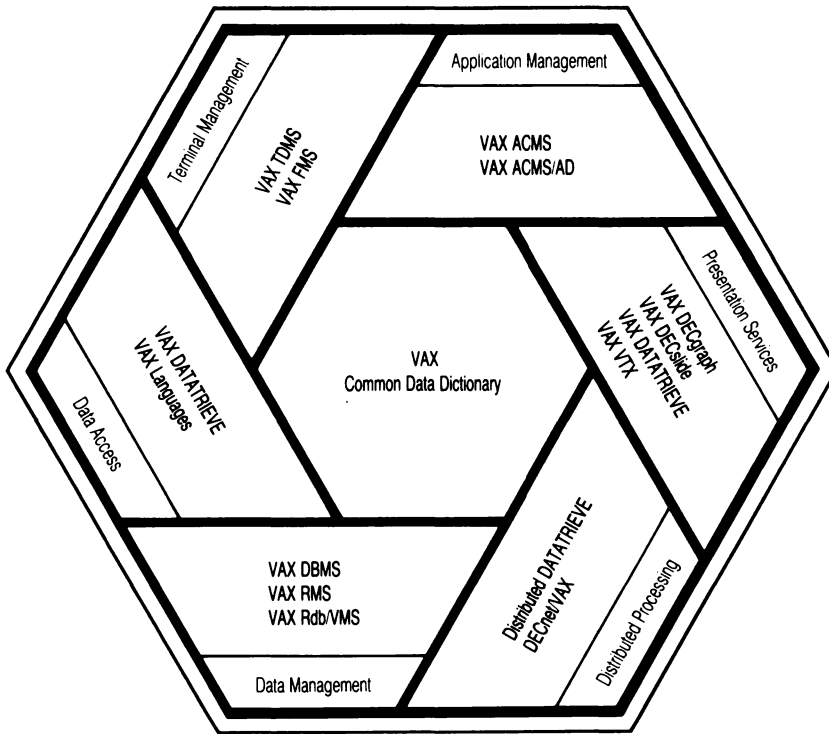


Figure 1-1 The VAX Information Architecture

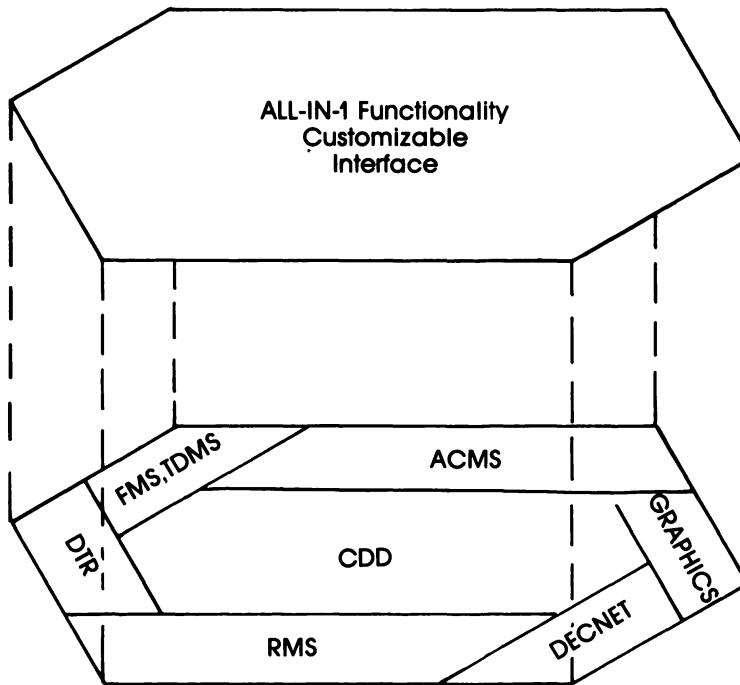
1.2 ALL-IN-1 in the VAX Information Architecture

The VAX Information Architecture is a unified approach to data organization, maintenance, manipulation, and retrieval. As illustrated in Figure 1-1, the architecture consists of modular software tools clustered around the Common Data Dictionary (CDD). Each of these tools can function independently, yet they are designed to work together. A central design feature of the VAX Information Architecture is the separation of form from function. Tools such as ACMS, FMS, and DATATRIEVE allow you to work with information formatted to your specifications. These high-level facilities are geared towards the form of the data, and they are complemented by other software tools in the architecture geared towards function. Other tools and components—the CDD, Record Management Services (RMS), and others—are geared towards manipulating and managing the data that comes from and goes to the user through the higher-level interfaces.

The separation of form from function is also a key element of the ALL-IN-1 Office Menu. ALL-IN-1 is an office-automation application of the VAX architecture.

1.3 ALL-IN-1 Architecture and Basic Concepts

ALL-IN-1 touches every component of the VAX Information Architecture polygon. As illustrated in Figure 1-2, you can combine and present the functionality of VMS to the user in many ways through ALL-IN-1. The ALL-IN-1 user interface is fully customizable, and the underlying functionality can comprise a wide combination of ALL-IN-1, VAX Information Architecture, and site-specific features.



MK-02005-00

Figure 1-2 ALL-IN-1 and the VAX Information Architecture

ALL-IN-1 implements a flow-control language of functions, form qualifiers and field qualifiers. Functions are the action commands of ALL-IN-1, and all processing revolves around them. FMS forms serve as the user interface throughout the system, and qualifiers associated with the form and its fields define how the interface looks and acts.

1.3.1 Form and Function

As mentioned earlier, ALL-IN-1 consistently separates form from function. Figure 1-3 illustrates this separation in the Word and Document Processing subsystem; the illustration also relates this separation to the differences between how ALL-IN-1 users and programmers view the system.

```

Rick Jones                                Fri 9-Nov-1984

Word and Document Processing Menu

      SEL Select                          Folder:
      C  Create                          Title:
      E  Edit                           Number:
      D  Delete
      P  Print
      I  Index

      S  Send by MAIL

      FC  File Cabinet
      DT  Document Transfer
      UD  User Defined Processing

Enter selection and press RETURN
  
```

```

1 Name .TYPE
MENU /CHOICE = CHOICE/CLEAR/DATE = DATE/MORE = "FC1,WPDXMENU"

2 Name .TYPE
/USER = USER/GET = DAY,

3 Name .TYPE
OA$DAY;CURFDR,OA$CURDOC__FOLDER OA$CURDOC__TITLE;

4 Name .TYPE
CURNUM,OA$CURDOC__DOCNUM/MAIL = MAIL

5 Name .TYPE
/HARD = "Word and Document Processing"/
  
```

Figure 1-3 Form and Function

Figure 1-3 illustrates the WP Form, an FMS form that serves as the interface to the Word and Document Processing subsystem. Named Data holds the form and field qualifiers that instruct ALL-IN-1 how to process the form and any information that a user may input. The functionality represented by Named Data is a combination of ALL-IN-1 function calls and form and field qualifiers.

Word Processing is one ALL-IN-1 subsystem. How subsystems interact depends primarily on:

- Named Data entries
- Form-library search rules
- Inherent menu-processing features

In summary, the potential of ALL-IN-1 arises from:

- The user-interface versatility and consistency, which allows a programmer to develop an application using one aspect of the interface, while a user can access that application through another aspect of the interface
- The ability of ALL-IN-1 to tap and integrate the entire resources of the VAX
- The ease with which ALL-IN-1 can be customized

The following section is an overview of the organization of ALL-IN-1.

1.3.2 How ALL-IN-1 Is Organized

For the general user, ALL-IN-1 is organized into the seven subsystems listed as the first seven options on the Main Menu. The primary user interfaces for these subsystems are the subsystem menus. More menus, entry forms, argument forms, and search forms provide detail and depth to the user interface.

A subsystem is a menu-based application. An ALL-IN-1 application is broadly defined as an application of ALL-IN-1 functionality.

A facility is a module of related ALL-IN-1 flow-control code and is an internal ALL-IN-1 concept.

For example, the Electronic Messaging subsystem (menu EMC or EMD and its associated forms) uses the MAIL application (the MAIL function and its subfunctions). You can customize any subsystem or you can create a new one, using the tools described in the APR. You can also build ALL-IN-1 subsystems around newly-written or pre-existing applications, regardless of the language or languages in which they are written.

If your system supports the BLISS-32 language, you can integrate your application into the ALL-IN-1 image. MAIL and CALENDAR are implemented in this way.

The following facilities and applications are documented in the APR:

- Initialization — Initializing the image
- Form Processing — Managing forms
- Field Processing — Managing fields within forms
- Functions — ALL-IN-1 verbs
- Subprocess
 Communications — Opening a window to the rest of VMS and
 communicating through it
- Symbols — Representing data
- File Cabinet — Managing personal and shared files
- Text Processing — Text editing and word processing
- Mail — User-to-user communication
- External
 Communications — Establishing links with other
 machines
- Time
 Management — Calendar and action items
- Help — On-line help
- Message — Creating and filtering error messages
- Script — Indirect command-file processing

2

ALL-IN-1 Initialization and Flow Control

2.1 Introduction

ALL-IN-1 is a user-mode program; it is a shared, native-mode executable image that uses form libraries and other RMS files. The image is stored in the ALL-IN-1 library directory, which is associated with the logical name OA\$LIB.

Before the ALL-IN-1 image associates with a DCL verb, the DCL SET COMMAND utility must be invoked. This is done during installation, to allow the ALL-IN-1 image to associate, by default, with the verb ALLIN1. This verb takes several qualifiers. When you invoke ALL-IN-1, it creates a tailored environment for the work session based partly on these command-line qualifiers and/or their default values. Initialization accomplishes the following:

- Validates the user
- Stores various privilege and flag bits
- Calls several routines that initialize various facilities (symbol table, form libraries, and so forth).

As the last step in initialization, ALL-IN-1 dispatches a FORM function call to display the initial form (the Main Menu is the default initial form). The input you supply through this form, and all subsequent processing in ALL-IN-1, occurs within a flow-control framework, a concept that is illustrated at the end of this chapter.

2.2 ALL-IN-1 Initialization

Following is a standard initialization sequence.

1 \$ ALLIN1

The image is activated with no qualifiers; all default values are implied. First, ALL-IN-1 determines whether the logicals OA\$LIB and OA\$DATA are already assigned; if so, it uses their equivalent names to identify the ALL-IN-1 library directory and the ALL-IN-1 data directory, respectively. If the logicals are not assigned, ALL-IN-1 assigns both of them to the directory where the image is located.

2 Open User Profile, OA\$DATA:PROFILE.DAT

The User Profile is the ALL-IN-1 accounting file, analogous to the VMS SYS\$SYSTEM:SYSUAF.DAT.

3 Determine Terminal Type

ALL-IN-1 queries the terminal to determine what type it is. ALL-IN-1 uses the terminal type to determine the mode in which to run. If the terminal can support FMS forms, ALL-IN-1 runs in Screen Mode. Otherwise, it runs in Hard Copy Mode.

However, if the Profile field SET__MODE is not blank, its contents determine the mode to run in and override any /TERMINAL value or terminal query answer-back.

4 Validate User

ALL-IN-1 reads the User Profile to determine whether you have a record in OA\$DATA:PROFILE.DAT. Then you may be prompted for an ALL-IN-1 username and/or password.

5 Create User's Environment

ALL-IN-1 reads your User Profile record to obtain the following context information:

- a Privileges: A series of 1-character privilege fields determines whether you can call functions interactively, enter DCL level, and so on.
- b Default device and directory: the User Profile identifies your default ALL-IN-1 user directory, and ALL-IN-1 assigns the logical OAUSER to that value.
- c Form libraries: ALL-IN-1 opens any form libraries you have identified in the Profile.
- d Special symbols: User Profile fields provide values for a number of special symbols. ALL-IN-1 references these symbols throughout the session (for example, when a subsystem needs the user name, editor style, and so on).

6 Permanent Symbol Table

The permanent symbol table, OAUSER:username.PST, is opened.

7 Display Starting Form

The final step in initialization is a call to the FORM function to display the starting form specified in the User Profile.

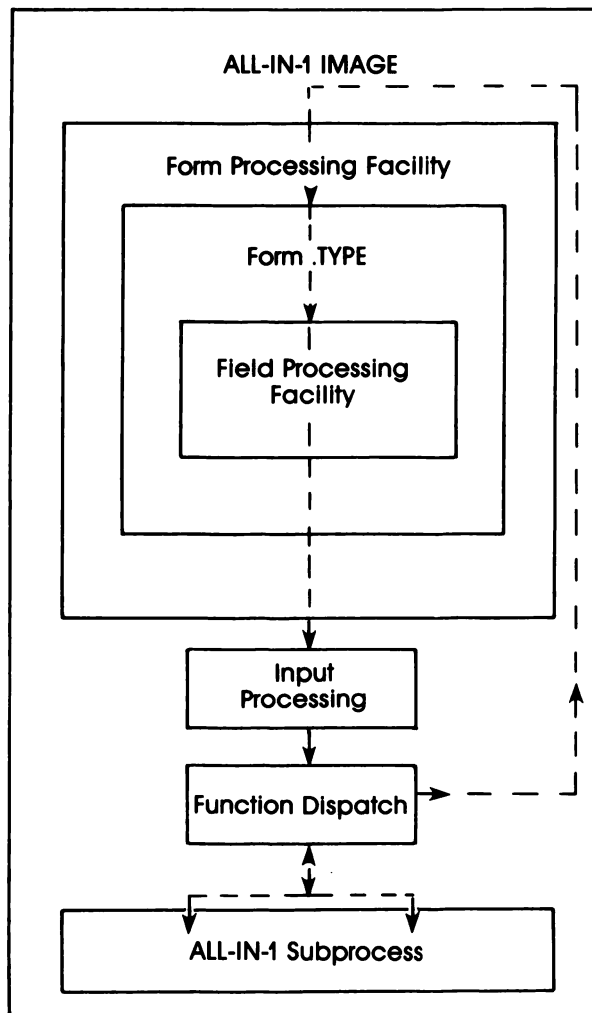
2.2.1 Partial Initialization

If you specify the /NOINITIALIZE qualifier on the command line when you call ALL-IN-1, it bypasses the preceding initialization process. Other programs that call ALL-IN-1 as a subroutine usually specify this qualifier. You can use other command-line qualifiers to further refine the initialization process.

2.3 ALL-IN-1 Office Menu Flow Control

Each ALL-IN-1 facility is a set of modules dedicated to a specific set of tasks. ALL-IN-1 function calls initiate processing within a facility. The term flow control refers to the sequence of events—function calls, facility actions and decisions, and other function calls—that occurs when ALL-IN-1 performs a task.

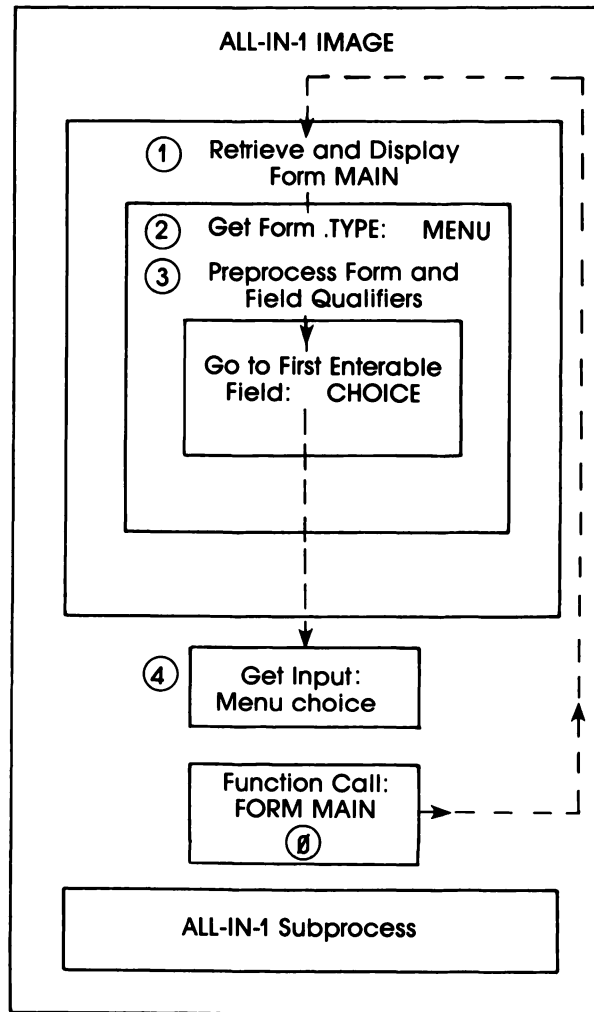
ALL-IN-1 operates within the framework illustrated in Figure 2-1.



MK-02006-00

Figure 2-1 Primary ALL-IN-1 Flow-Control Framework

When the initialization sequence concludes, the initial form displays.
Figure 2-2 illustrates the flow-control pathway when the initial form is the default form, MAIN.



MK-02003-00

Figure 2-2 Flow Control Path: Display Main Menu (Cont.)

- Number on Figure 2-2 Identifies the following action:
- ① Function calls control the flow of processing.
 - ① Form Processing locates the form specified by the FORM function.
 - ② Form Processing identifies the form .TYPE, which determines how the form is processed.
 - ③ Form and Field Processing direct the preprocessing of appropriate form and field qualifiers.
The form displays.
 - ④ Input Processing, a transparent facility, determines the source of the next input (for example, interactive user input or a script line).

This flow pattern applies primarily to the display and processing of FMS forms, the standard ALL-IN-1 user interface. Flow control varies depending on the function that is currently being dispatched, but this framework remains consistent.

Figure 2-2 Flow Control Path: Display Main Menu

Form Processing

3.1 Introduction

ALL-IN-1 uses VMS Forms Management System (FMS) forms as the interface between a user and the routines that perform the tasks that the user requires. ALL-IN-1 forms are screens or screen sets designed and displayed using functionality provided by FMS and flavored by the ALL-IN-1 user interface. The ALL-IN-1 Form Processing facility controls form display and general form processing.

A form consists of static background text and variable fields. You add background text and assign form-wide display attributes when you create a form. Fields display and/or receive variable information, and you assign field attributes when you create the form. Named Data is additional information that you supply and that ALL-IN-1 uses as field and form processing instructions.

The **FORM** function displays forms. ALL-IN-1 uses forms that are stored in form libraries, which are described in the following section.

The *User's Reference* and the on-line Help files provide general guidelines on how to manage forms within the ALL-IN-1 Forms Development subsystem. The chapter on Application Development Tools and Design Guidelines, gives a more detailed description of how to create and manage forms within ALL-IN-1. Refer to the FMS reference manual set for detailed instructions on how to edit forms.

3.2 Form Libraries

A library is a special type of VMS file that contains modules of text, forms, binary object code, or help text and contains its own index to those modules. A form library is a single VMS disk file that contains one or more forms. Using an embedded index, a given form may be directly accessed by its name. In ALL-IN-1, the common form library, OAFORM.FLB, resides in the OA\$LIB directory and contains most of the forms that serve as the general user interface. Another form library, OA\$LIB:MEMRES.FLB, contains memory-resident forms that ALL-IN-1 can access rapidly. MEMRES contains forms such as MAIN, the Main Menu; OA\$VIEW_FIELDS, which displays when a user presses Gold V; form set PROFIL, which defines the User Profile; and others. Since memory-resident forms are linked directly into the ALL-IN-1 image, ALL-IN-1 treats MEMRES.FLB differently than other form libraries.

Another small form library, OA\$LIB:STANDARD.FLB, contains prototype forms that you can use as models for creating custom forms.

System Manager forms are stored in a separate library, OA\$LIB:MANAGER.FLB. These forms are described in the *System Manager's Guide*.

In addition, when the system manager establishes a new user, ALL-IN-1 places an empty form library, USER.FLB, in that new user's default ALL-IN-1 directory. Use USER.FLB to store customized and/or newly created forms that are to be used privately by a particular user.

When system-wide forms are modified, replace them in the library in which they originated. For example, replace modified MEMRES forms in the OA\$LIB:MEMRES.FLB library; replace modified OAFORM forms in the OAFORM.FLB library. Store new application specific forms in a new form library.

The following section describes how ALL-IN-1 searches the form libraries.

3.2.1 Searching Form Libraries

The Form Processing facility is responsible for searching form libraries when a form is to be displayed. OA\$LIB:MEMRES and OA\$LIB:OAFORM are the default libraries; all users automatically have access to them. ALL-IN-1 always searches MEMRES first and usually searches OAFORM last. The search finds the first occurrence of the form.

To access forms in other libraries such as OA\$LIB:MANAGER or OAUSER:USER, enter the library name (less the ".FLB" extension) in the Form Libraries field (FRMLIB) of the User Profile entry form set, PROFIL. Add the directory prefix if it is different from your OAUSER default directory. ALL-IN-1 searches MEMRES first and then the user-specified form libraries in the order they are entered in field FRMLIB. ALL-IN-1 searches OAFORM last. To direct Form Processing to search OAFORM in some other order, enter OA\$LIB:OAFORM in field FRMLIB to override its default position as the last library searched.

You cannot display ALL-IN-1 forms that are not associated with a form library through the Form Processing facility.

3.3 The FORM and SHOW__FORM Functions

3.3.1 The FORM Function

The FORM function invokes the Form Processing facility to get, preprocess, display, and postprocess a form. You can use the FORM function to call a form as it was originally created, or to redefine some aspects of the form within the function call.

The FORM function has the following syntax:

FORM [form-name [type] [/qualifier[= arg] [/qualifier[= arg]...]]

where

form-name is the optional name of the form to process.

type is the optional form .TYPE. The form type corresponds to the .TYPE data element in Named Data. Specifying a type parameter in a function call resets the .TYPE directive and the form qualifiers that the form was created with and allows the form to be used as a different interface.

`/qualifier=arg` are one or more optional form qualifiers, some of which may have arguments. These qualifiers will augment the existing form qualifiers in Named Data or replace them, depending on the individual qualifier.

When you omit the form name, the FORM function acts as a display/ prior menu function. A substitution character (.) is also a valid FORM parameter, and it is interpreted as an instruction to display the current or most recently processed form. Specify the substitution character in place of the form name:

`FORM.[type][/qualifiers]`

where the period refers to the currently displayed or most recently processed form. As with other form calls, the form-type parameter is optional when the substitution character is used, allowing the current form to either be reprocessed as the same form type with new or additional form qualifiers or be processed as a new form type with new qualifiers.

You must supply either a form name or the substitution character if you are specifying a new form type and/or form qualifiers.

There are many possible ways to combine the form name, a form type and form qualifiers in a FORM function call.

3.3.2 The SHOW__FORM Function

SHOW__FORM displays any standard or nonstandard form. No fields are activated. The form remains on the screen until you specify another function, usually with the /POST__FUNCTION form qualifier. This changes the screen.

This function is often used to display forms in script-driven demonstrations and Computer Based Instruction (CBI).

3.4 ALL-IN-1 Named Data Conventions

FMS provides a convenient means of associating additional non-FMS information with a form; this mechanism is called Named Data. FMS Named Data is an array of from 0 to about 65,000 elements, each with two fields: Name and Data. The name is a 30-character field, and the data is 80

characters. If the data entry is longer than 80 characters, it may be extended by repeating the name entry in another Named Data element and continuing the associated data field.

Named Data is managed, but not interpreted by, FMS. ALL-IN-1 takes advantage of Named Data by defining several types of Named Data entries:

- Form-type directives
- Form-set directives
- Menu options
- File-definition directives
- Keyboard definitions
- Field names
- Named Data chain directives

This chapter describes all the types of Named Data entries except keyboard definitions and field qualifiers, which are covered in the chapter on Field Processing. The Application Development Tools and Design Guidelines chapter discusses conventions of form appearance.

Although ALL-IN-1 syntax allows any one form to be used in several ways, most forms serve a single purpose, such as presenting menu options or accepting information for a data-set search. ALL-IN-1 groups forms into functional categories known as form types (or form `.TYPES`). Forms that are the same `.TYPE` are processed by the same routines in the Form Processing facility. Nonstandard forms have no form `.TYPE`.

3.4.1 ALL-IN-1 Form `.TYPES`

The Named Data of all standard forms contains a required `.TYPE` directive that tells the Form Processing facility how to process the form. A form `.TYPE` directive consists of the name entry `.TYPE` and a corresponding data entry that describes the form `.TYPE`. Some special forms are processed as unique form `.TYPES`; that is, some form types comprise a small, fixed number of special forms. The form `.TYPES` are:

- ARG — A form type that is used to accept user input as arguments, possibly to be passed to command procedures or scripts or saved in symbols.

- **CALC** — A form type that is used for repetitive actions.
- **CLEAR** — A form type that clears the screen (Form OA\$CLEAR).
- **DCL** — A special form type, in which the user enters DCL Mode in a subprocess. This results in a “\$” entry or a DCL function call.
- **DESK** — A special form type that emulates a simple electronic calculator. The Desk Calculator form, OA\$DESK, is the DESK form.
- **DTR** — A special form type which allows commands to be input to DATATRIEVE (Form OA\$DTR).
- **EDIT** — A special form type that allows text editing. Form OA\$EDIT, which is memory-resident, is the EDIT form and displays when the EDIT function is called.
- **ENTRY** — A Data entry form which accesses data files.
- **HELP** — A special form type that displays Help text. Forms OA\$HELP__TOP and OA\$HELP__BOTTOM, which are memory-resident, are the HELP forms, and one of them displays when the HELP function is called.
- **LIST** — A special form type that outputs text to the screen and displays when the LIST function is called (Form OA\$LIST).
- **MENU** — A menu form that provides a list of options and an area where a choice can be entered.
- **SEARCH** — A form type that is used to search a data file and build a selection list.
- **SELECT** — A form type that is used to build, display, and allow selections from a collection of records.
- **VIEW** — A special form type that displays the current context information. The view forms, OA\$VIEW__FIELDS, OA\$VIEW__MESSAGES, and OA\$VIEW__ND, are memory resident and are defined in form DEFAULT to display as a result of Gold-key combinations.

The following special form .TYPEs — DCL, DTR, EDIT, HELP, LIST, and VIEW — are either memory-resident forms or internal constructs that are processed in the same way as other forms. These forms are not covered in detail as form .TYPEs in this chapter. See the related DCL, DTR, EDIT, HELP, and LIST functions in Chapter 6. VIEW forms are called by default through Gold-key sequences.

The field-oriented form **.TYPEs** are **MENU**, **ENTRY**, **ARG**, **CALC**, **DESK**, **SEARCH**, and **SELECT**.

3.4.2 The **.MORE** Directive

The **.MORE** directive provides a way to extend the number of entries in Named Data of any form type.

The syntax of **.MORE** is:

```
n Name .MORE  
form-name
```

where

form-name is the name of another form which contains additional Named Data entries.

n is the number of the Named Data entry.

3.4.3 Searching for Named Data

When **ALL-IN-1** looks for instructions on interpreting input, it searches in the following order:

- 1 The Named Data of the current form is searched first.
- 2 If the Named Data is part of a form set, the additional pages in the form set are searched next.
- 3 If a **.MORE** is present on the form and **ALL-IN-1** does not find the information that it needs on that form, it searches the Named Data of the form to which the **.MORE** points. In the case of menu forms, a **.MORE** chain should result in a circle of forms. Thus, the last form in the chain should point back to the first form. This allows the user to be in any form in the chain and access any option within the chain. If the forms were not in a circle of forms, the only valid option would be those forms that are down the chain.
- 4 If a **/MORE** is present **ALL-IN-1** searches those forms which are qualified by the **/MORE**. If **/CAPTIVE** is specified, the search ends.
- 5 The Named Data of form **DEFAULT** (or the current default form specified by the **SET_MENU** function) is searched last.

3.4.4 Form Qualifiers

The form `.TYPE` directive may be qualified with form qualifiers, and, in all field-oriented form `.TYPE`s, individual fields may be entered in Named Data and qualified with field qualifiers. The general syntax for form and field qualifiers is the same:

`/qualifier[ = arg][/qualifier[ = arg]`

where

`“/”` is required syntax.

`qualifier` is any valid form or field qualifier.

`=arg` is one or more individual qualifier arguments, all of which may be optional. (Some qualifiers require single or double quotes around the argument or arguments.)

Form qualifiers in Named Data have the following general syntax:

Named Data

1 Name `.TYPE`

`form-type/qualifier[ = arg][/qualifier[ = arg]...]`

3.4.4.1 Form Qualifiers In Function Calls. You may specify a form qualifier in a FORM function call. How the qualifier interacts with existing qualifiers depends on several things, including whether the qualifier is already present and active, how many arguments (if any) it takes, and any special side effects of the qualifier. The most important factor, however, is whether you also specify a form `.TYPE` in the FORM call. When you call a form as a new type, ALL-IN-1 ignores all the original form qualifiers.

When you call a form as the same type but with a different combination of qualifiers, ALL-IN-1 processes the qualifiers identically unless:

- A qualifier in the new call explicitly augments or overrides an original qualifier.
- OR
- All or some of the original qualifiers are reset with the `/RESET` qualifier in the new call.

When you specify a form qualifier in a FORM function call, ALL-IN-1 usually dynamically appends the qualifier to the existing qualifier string specified in Named Data. If the qualifier is already present and active, it usually replaces itself. The exception is /RESET, which causes all preceding qualifiers to be ignored and is thus a special case. For example, form MENFRM could contain the following Named Data entries:

Named Data

1 Name .TYPE

MENU/CHOICE = CHOICE/CLEAR/DATE = DATE/USER = USER/TITLE = TITLE

The following function call causes CHOICE2 to replace CHOICE as the choice field and causes all enterable fields on the form to be highlighted:

<FORM MENFRM/CHOICE = CHOICE2/HIGHLIGHT

All other qualifiers remain unaffected and are processed as before.

When a form qualifier that takes more than one argument is specified in a FORM function call, the qualifier is usually dynamically appended to the existing qualifier list stored in Named Data (or appended from the previous FORM function call). If the new qualifier is already present and active, its arguments are usually used to augment the existing argument list. The exceptions are:

- /FIELDS, which redefines the set of enterable fields on the form and thus replaces, rather than augments, itself. /FIELDS may also affect the behavior of other field qualifiers, such as /BEGIN. See the individual form qualifiers for a more detailed discussion.
- /PRE_FUNCTION and /POST_FUNCTION, which specify a function or list of functions to execute either before or after the form displays. /PRE_FUNCTION and /POST_FUNCTION also replace themselves: that is, the function list specified as the /PRE_FUNCTION or /POST_FUNCTION argument replaces the function list specified by any previous qualifier calls.
- /MORE, which is a menu form qualifier that specifies additional forms whose Named Data is to be appended to the current menu's Named Data. /MORE replaces, rather than augments itself.
- /GET, which always appends to itself and is not cleared by the /RESET qualifier. However, individual fields in the /GET list are replaced.

For example, form WP contains the following information in Named Data:

Named Data

1 Name .TYPE

MENU/GET = CURFDR,\$WPFDR;CURNUM,OA\$CURDOC__DOCNUM

The /GET retrieves the value of permanent symbol \$WPFDR and places it in field CURFDR; it also retrieves a file-cabinet document number and places it in field CURNUM.

The following function call overrides the /GET on field CURFDR. The /GET on field CURNUM remains unchanged.

<FORM WP/GET = CURFDR,OA\$USER

In the following example, form WP is called as an argument form, and all Named Data qualifiers are overridden.

<FORM WP ARG/GET = CURFDR,OA\$USER

NOTE: *Because Field Processing, rather than Form Processing, is in control when a particular field is entered, field qualifiers overrule form qualifiers when both apply. The results of specifying overlapping form and field qualifiers are discussed under the particular form/field qualifier.*

The common, or shared, form qualifiers are covered in the following section. Field qualifiers are described in detail in the chapter covering Field Processing.

3.4.5 Common, or Shared, Form Qualifiers

The form qualifiers listed in this section are valid for any field-oriented form. Some common form qualifiers are processed slightly differently by different form types. For further information, see the sections in this chapter on Menu forms, Entry forms, Argument forms, Calc and Desk forms, Search forms, and Select forms.

/BEGIN = field

The /BEGIN form qualifier has a single, required argument: the name of the starting field when the form displays. The field must be enterable. If it is not, the next enterable field following the specified field is used. After

the user exits from this field, the other enterable fields are touched in the order specified when the form was created. /BEGIN does not restrict the user from either backspacing to a previous field or tabbing to subsequent fields.

When /BEGIN is not specified, the user is placed in the first enterable field on the form.

For example:

Named Data

1 Name .TYPE
ARG/BEGIN=FLDA

NOTE: *In menus, /BEGIN is ignored unless the /FIELDS qualifier explicitly activates the beginning field. /BEGIN is also processed differently in entry forms.*

/CLEAR[= field[,field...]]

The /CLEAR form qualifier clears the specified field or fields just before the form displays. /CLEAR is valid for all form types except entry forms. On Entry forms it conflicts with entry-form preprocessing because the fields are automatically cleared when the form displays.

/CLEAR is usually specified with at least one field-name argument. It works with no arguments in one specialized case: /CLEAR with no arguments in Named Data clears the choice field, but only when /CLEAR follows the /CHOICE qualifier. A /CLEAR that precedes /CHOICE in a menu or a /CLEAR with no qualifiers in other form .TYPES is ignored. To clear other menu fields in addition to the choice field, all fields must be specified.

For example, the CHOICE, PROMPT, and ARG1 fields of the following menu clear when the form is preprocessed:

Named Data

1 Name .TYPE
MENU/CHOICE = CHOICE/CLEAR = ARG1,CHOICE,PROMPT

Form Processing

/DATE = field

The **/DATE** qualifier specifies a field that receives and displays the current date. The field should be at least 9 characters long and created as an alphanumeric field.

For example:

Named Data

1 Name .TYPE
SEARC/CLEAR = FIELD1/DATE = DATE

/DATE is equivalent to **/GET=field,OA\$DATE__FULL** and is provided as a convenience.

/DISPLAY[= function[\function...]]

The **/DISPLAY** qualifier marks a form to be processed as display-only. The optional argument is one or more **ALL-IN-1** functions that execute after the form displays and is processed.

/DISPLAY is similar to the **SHOW__FORM** function, but where **SHOW__FORM** simply displays the image of a form on the screen, **/DISPLAY** executes all specified pre and postprocessing of the form.

Although the function-string argument is optional, if it is omitted the form never actually displays; it must be forced to remain on the screen with a postprocessing **PROMPT** or similar function call.

For example:

```
<FORM SCHEDM/DISPLAY/POST__FUNCTION = "IFEXIT\PROMPT Press Exit Screen"
```

This interactive call works on **ARG** and **ENTRY** forms, but not menus or **OA\$DESK**. It displays the form and then displays the specified prompt on line 24 if the user leaves the form in the typical manner. The following call is similar, but not identical:

```
<FORM SCHEDM/DISPLAY = "PROMPT Press Exit Screen"
```

Multiple functions can be stacked within the **/DISPLAY** argument. For example, assuming forms **A**, **B**, and **C** are present:

```
<FORM OA$DESK/DISPLAY = "SHOW__FORM A\SHOW__FORM B\SHOW__FORM C"
```

/DISPLAY is useful to create demonstrations and **CBI** lessons.

`/FIELDS = field[,field...]`

The `/FIELDS` form qualifier takes as arguments the names of one or more enterable fields that are to be activated while the form displays. All enterable fields not specified in the `/FIELDS` qualifier are ignored during field processing. Form qualifiers that process display-only fields are not bypassed. This is covered later in the chapter.

By default, field processing processes all field qualifiers and solicits input from all enterable fields. The `/FIELDS` qualifier effectively redefines the form, creating a subset of fields that can be entered.

If the `/BEGIN` form qualifier is present, that beginning field must be among those activated by `/FIELDS`.

NOTE: *`/FIELDS` is processed slightly differently by menus and entry forms.*

For example, the following Named Data entry is associated with form ARGFORM:

Named Data

1 Name .TYPE

ARG/GET = ACCOUNT,"ACCOUNT.IDNUM[\$NAME]"/HIGHLIGHT/CLEAR = THIRD

This argument form is called with any subset of its enterable fields:

`<FORM . /FIELDS = FIRST,SECOND,FIFTIETH/BEGIN = FIFTIETH`

Fields `FIRST`, `SECOND`, and `FIFTIETH` are activated, and field `FIFTIETH` is the first field the user enters. Those activated fields that are enterable are highlighted during preprocessing, but field `THIRD` is not cleared during preprocessing because it is not activated.

Although a field may not be activated by `/FIELDS`, `ALL-IN-1` may still modify it during form processing if there are form qualifiers that refer to the bypassed field. Thus, in the preceding example, field `ACCOUNT` is always loaded with the value of the specified DSR because the `/GET` field qualifier is processed before `/FIELDS`.

`ACCOUNT` is processed like a display-only field for the duration of this `FORM` call; similarly, fields that are defined as display-only are always activated regardless of whether they are identified by `/FIELDS` since they are processed at the Form Processing level.

The order in which the fields are specified with **/FIELDS** is unimportant; you identify the order of field access through FMS when you create the form.

/GET = field,symbol[;field,symbol...]

The **/GET** form qualifier loads the specified field or fields with the value of the associated symbol when the form is preprocessed. The field may be any valid field with any combination of compatible FMS attributes on the qualified form. Symbol-value/field compatibility is one of the few restrictions on **/GET**—that is, if the symbol to get contains an alphanumeric value and the field to load has been specified as numeric-input-only (or vice-versa), the **/GET** will be unsuccessful. The symbol may be any valid **ALL-IN-1** symbol or symbol combination.

/GET is processed after the **/PRE_FUNCTION** form qualifier; thus, symbols that are assigned values through the function called by **/PRE_FUNCTION** may be referenced by **/GET**.

Also, as covered under **/FIELDS**, **/GET** is processed before **/FIELDS**, so a **/GET** always retrieves a value into a field regardless of whether that field is activated by **/FIELDS**.

Examples of the qualifier are given below:

Named Data

1 Name .TYPE

MENU/DATE = DATE/GET = DOCNAM,\$EDITDOC;

2 Name .TYPE

DATE,OA\$DATE;REVDAT,"'Last Revised: ' DOCDB.MODDAT[OA\$CURDOC]"

In this example, three types of symbols — permanent symbol **\$EDITDOC**, special symbol **OA\$DATE**, and a combination of a literal string with data file reference **DOCDB.MODDAT[OA\$CURDOC]** — are retrieved and loaded into fields in the form.

The **/GET** form qualifier is closely related to the **GET** function and to the **/GET_SAVE** field qualifier. For example, assume menu form **EXMPL**

contains an enterable field named **TITLE**. If **EXMPL**'s Named Data contains the form qualifier:

```
/GET = TITLE, OA$TITLE
```

then the user's title automatically loads into the field and displays when the form is called. The combination:

```
/PRE__FUNCTION = 'GET TITLE = OA$TITLE'
```

where **/PRE__FUNCTION** is a form qualifier that calls the **GET** function, is nearly identical. The only difference is when the **GET** is performed, as described previously.

If you make the following interactive function call from the choice field:

```
<GET TITLE
```

the value of field **TITLE** displays on line 24.

Assume that **TITLE** is an enterable field qualified by the following field qualifier:

Named Data

```
1 Name TITLE  
/GET__SAVE = OA$TITLE:5
```

as well as by the **/GET** form qualifier:

```
2 Name .TYPE  
MENU/CHOICE = CHOICE/GET = TITLE,$CURTITLE
```

In this case, when the user enters the field, and triggers Field Processing, the **/GET__SAVE** overrides the **/GET**, and a 5-character substring of **OA\$TITLE** displays. For this reason, **/GET** and **/GET__SAVE** should usually be treated as mutually exclusive. Because **/GET__SAVE** overrides the **/GET** when field processing touches that field, the results may be confusing.

The major difference between **/GET** and **/GET__SAVE** is that the **/GET** qualifier is always processed, while **/GET__SAVE** is only processed for fields which are active. For this reason **/GET** is usually used on menus to load the context fields.

One other difference between /GET and /GET__SAVE or the GET function is illustrated in the following example. The quotes around the data set reference (DSR) are not required for the GET function or for /GET__SAVE; in fact, quoted strings are treated as literals by GET and /GET__SAVE. Quoted strings may be specified as the key for any DSR, but with /GET, an alternate quote character is used to enclose the key value. For example:

1 Name .TYPE

MENU/CHOICE = CHOICE/GET = AUTHOR,"DOCDB.AUTHOR['Monthly Report']";

2 Name .TYPE

DATE,OA\$DATE;REVDAT,"DOCDB.MODDAT[OA\$CURDOC]";FOLDER,\$CURFDR

3 Name TITLE

/GET__SAVE = PROFIL.TITLE["George Herman"]

The /GET in Named Data entry 1 retrieves the author of the document filed under the title "Monthly Report" and puts the name in field AUTHOR.

/HARD = "description"

The /HARD qualifier is used to display information about a form while in Hard Copy Mode. Since FMS forms do not display on hardcopy terminals, the concepts of field and form are not very meaningful. When ALL-IN-1 is run in Hard Copy Mode, the user must try to navigate without screen displays of menu options and other helpful information. The /HARD qualifier provides a line of information that can help guide the user through the system.

For example:

Named Data

1 Name .TYPE

MENU/CLEAR/HARD = "The DATATRIEVE Menu: Enter AB, CD, or EF"

The /HARD field qualifier performs the same function for fields.

/HELP = keyword-list-symbol

The **/HELP** form qualifier has a single argument: an ALL-IN-1 symbol representing a list of help keywords. These keywords identify help text to display. Help is largely form-oriented, and the Help Facility uses the name of the current form to locate help text. The **/HELP** qualifier overrides this process and is most useful when one help topic applies to more than one form.

For example:

Named Data

1 Name .TYPE

MENU/CHOICE = CHOICE/DATE = DATE/HELP = PROFIL.MAIL__MENU[OA\$USER]

Data set reference PROFIL.MAIL__MENU[OA\$USER] is translated before the help library is searched, and the keyword it contains (that is, the name of the top-level Messages menu) is used to retrieve help text. The text displays when the user presses Gold H in the CHOICE field of this menu.

From within a command procedure:

\$! +

\$! COMMON__FORM is a DCL symbol that contains the name

\$! of a form whose help is to be displayed for NEWFORM

\$!-

\$ WRITE OAMAILBOX "OA FORM WPLPSEL/HELP = " " "COMMON__FORM' " " "

\$ @DCLMAILBOX:

/HIGHLIGHT

/HIGHLIGHT highlights all enterable fields that are participating on the form with reverse video. When present, the **/FIELDS** form qualifier identifies the participating fields; when **/FIELDS** is not specified, all enterable fields are highlighted.

When one or more of the four FMS Field Highlight attributes are specified and **/HIGHLIGHT** is not present, FMS will highlight the current field (the

field containing the cursor). Thus, **/HIGHLIGHT** is used in conjunction with FMS to more clearly identify the status of fields on a form. For example:

- FMS video attributes could be used to permanently **UNDERLINE** all fields.
- **/HIGHLIGHT** could be used to highlight enterable fields with reverse video.
- FMS Field Highlights, assigned as Form Attributes, could be used to **BOLD** the current field.

For example:

Named Data

```
1 Name .TYPE  
ENTRY/MODE = ADD/FIELDS = F1,F2,F3,F4,F5/HIGHLIGHT
```

NOTE: */HIGHLIGHT and FMS Field Highlights may have a noticeable impact on system performance when used on large forms or on a large number of forms.*

/KEYPAD

The **/KEYPAD** qualifier deactivates the keypad. If the qualifier is not present, the keypad is activated. The **/KEYPAD** qualifier provides the same functionality as the **KEYPAD** function. But, if the **/KEYPAD** qualifier is present on a form when the **KEYPAD** function is called, the **/KEYPAD** qualifier overrides the **KEYPAD** function. Thus, if pre-processing or post-processing of a form call the **KEYPAD** function, the effects of the function are cancelled when the form is processed.

For example:

Named Data

```
1 Name .TYPE  
ARG /PRE = 'KEYPAD OFF\COMMAND A\KEYPAD ON\COMMAND B'/KEYPAD
```

In this example, the keypad is turned off during the command procedure A, turned back on for command procedure B, and turned off when the form requests user input.

/LOG = "text"

/LOG sends the quoted text argument to a log file when logging is enabled for the user.

For example:

Named Data

1 Name .TYPE
ARG/LOG = "Reached form ARGFRM"

/MAIL = field

The **/MAIL** qualifier specifies a field to receive and display the number of outstanding mail messages waiting for the user. The field should be between 21 and 23 characters long and created as an alphanumeric field.

For example:

Named Data

1 Name .TYPE
ENTRY/MAIL = MAIL

/MAIL is equivalent to **/GET=field,OA\$MAIL_COUNT_DISPLAY** and is provided as a convenience.

/NEXT = form

The **/NEXT** qualifier specifies a form that displays when the user exits the current form in any way. **/NEXT=form_name** is equivalent to **/POST_FUNCTION='FORM form_name'** and is provided as a convenience.

For example, the Named Data of menu NOWFORM contains the following entry:

Named Data

1 Name .TYPE
ARG /NEXT = NEXTFORM

When the user presses RETURN or EXIT SCREEN in form NOWFORM, NEXTFORM displays.

Like the `/POST__FUNCTION` form qualifier, `/NEXT` overrides the `EXIT__ SCREEN` key. The `/NEXT`-specified form is called after the current form exits and before form/field postprocessing begins.

`/NEXT` may be used to allow users to bypass one or more forms when they want to exit an application in which those forms have become stacked up.

`/OVERLAY[ = form[,form...]]`

Unless the screen is explicitly cleared first, all forms that are less than the full 23-line (VT-series) screen length overlay, by default, whatever is on the screen. The `/OVERLAY` qualifier prevents this full-screen repaint; when the form exits, only the occluded lines of the underlying form, and any other lines that changed as a result of broadcast messages or prompts, restore.

Often an overlay form, particularly an overlay menu, needs to be displayed in conjunction with one particular underlay form in order to be meaningful. For this reason, there is an optional argument on `/OVERLAY`: the name of one or more forms that must underlie the overlay when the overlay displays.

For example, form `MAIN2`, an overlay extension of the Main Menu, contains the following Named Data entry:

Named Data

```
1 Name .TYPE  
MENU/OVERLAY = MAIN
```

Since one or more forms can be, and often are, invoked from form `MAIN2`, and similar overlay forms, an `EXIT SCREEN` that backs up to the overlay may cause that overlay form to display in some confusing way.

For example, suppose form `MAIN2` is current, and option `US` is chosen. If form `MAIN2` Named Data simply contains `/OVERLAY` with no argument, then when `EXIT SCREEN` is pressed from form `US`, `MAIN2` overlays `US`. The options displayed on form `MAIN2` can still be chosen, but the screen display gives the misleading impression that the `MAIN2` options are associated with the User Setup menu.

`/OVERLAY` is not meaningful on full-screen forms.

NOTE: A similar task is performed by *CLEAR* when this function is supplied with the optional form parameter, and that form is an overlay. In this case, just the occluded portion of the screen is refreshed — any broadcast messages and other changes to the rest of the screen remain.

`/POST__FUNCTION='function [\function...]`

The `/POST__FUNCTION` qualifier specifies one or more functions to perform during form postprocessing after the user exits the form in any way. If you specify a function list, separate function calls from one another with backslashes.

Also, when you specify a function list, enclose the entire list in single or double quotes. Complex function lists often include one or more combinations of quoted strings, so the choice of quote characters both within and around the function list becomes important. Quotes are optional with single `/POST__FUNCTION` functions.

When a form displays or is active one time only, *ALL-IN-1* invokes the function(s) directly after the form exits. When a form cycles until dismissed by the user, *ALL-IN-1* invokes the function(s) after each cycle.

For example:

Named Data

1 Name .TYPE

`ARG /POST__FUNCTION='IFEXIT\DTR SET LIBRARY CDD$TOP.ADDRESSES'`

In this example, *ALL-IN-1* calls the *DTR* function if the user leaves the argument form normally.

NOTE: Unlike the `/POST__FUNCTION` field qualifier, *ALL-IN-1* executes the `/POST__FUNCTION` form qualifier even when the user presses *EXIT SCREEN*, unless the form is an *ENTRY* form. (In this case, the `/POST__FUNCTION` qualifier is only executed when the form is successfully filled in.) For this reason, you should precede all function calls in a `/POST__FUNCTION`-specified list with *IFEXIT* whenever you want the functions to be bypassed if *EXIT SCREEN* is pressed.

`/PRE__FUNCTION='function [\function ...]'`

The `/PRE__FUNCTION` form qualifier identifies one or more functions to perform during form preprocessing. If you specify a function list, separate function calls from one another with backslashes.

Also, when you specify a function list, enclose the entire list in single or double quotes. Complex function lists often include one or more combinations of quoted strings, so the choice of quote characters both within and around the function list becomes important. Quotes are optional with single /PRE_FUNCTION functions.

When a form displays one time only — for example, a MENU, ARG, SEARCH, SELECT, or VIEW form, or an ENTRY form with /ONCE — ALL-IN-1 invokes the function(s) just before the form displays. When a form cycles until the user dismisses it — for example, a CALC form, or an ENTRY form without /ONCE — ALL-IN-1 invokes the function(s) before each cycle.

For example:

Named Data

1 Name .TYPE

```
ARG / PRE='COMMAND PREPROCES \CLEAR' / FIELDS=A,B,C
```

In this example, the command procedure PREPROCES executes just before the form displays; when PREPROCES exits, the CLEAR function clears the screen. /FIELDS is processed as a form qualifier.

/RESET

The /RESET qualifier causes ALL-IN-1 to ignore all the currently active form qualifiers. This qualifier is only meaningful in FORM function calls and is not intended as a permanent Named Data qualifier.

For example, if form EM contained the following Named Data entries:

Named Data

1 Name .TYPE

```
MENU / PRE="COMMAND PREPRO" /CHOICE=CHOICE/CLEAR/GET=MSG,$MSG
```

then calling this form with a /RESET qualifier causes form processing to ignore all previous qualifiers (except the /GET qualifier):

```
<FORM EM/RESET/CHOICE=CHOICE2
```

/SUPER[= username-symbol]

FMS allows fields to be given a supervisor-only attribute during creation of a form. When any supervisor-only fields are present on a form, the **/SUPER** qualifier restricts access to these fields to the individual identified by the username-symbol argument. This argument may be an ALL-IN-1 symbol-table symbol or a quoted string.

/SUPER overrides the PRVSUP privilege in the User Profile, which allows a user to access all supervisor-only fields on all forms.

For example:

Named Data

```
1 NAME .TYPE  
ENTRY /DATA = PROFIL /SUPER = $ONEUSER /MODE = UPDATE
```

When this form displays, only the user who is identified by the permanent symbol **\$ONEUSER** is able to access the supervisor-only fields.

NOTE: *If users have the PRVCMD privilege, they can easily override /SUPER by simply calling the form interactively and providing their own name as the supervisor. For example:*

```
<FORM ANYFORM/SUPER = "KLANBURG"
```

/TITLE = field

The **/TITLE** qualifier specifies a field to receive the user's title.

For example:

Named Data

```
1 Name .TYPE  
MENU/CHOICE = CHOICE/CLEAR/TITLE = TFIELD
```

/TITLE is equivalent to **/GET=field,OA\$TITLE** and is provided as a convenience.

`/USER = field`

The `/USER` qualifier specifies a field to receive the user's full name. For example:

Named Data

1 Name .TYPE
MENU/CHOICE = CHOICE/CLEAR/USER = USER

`/USER` is equivalent to `/GET = field,OA$USER` and is provided as a convenience.

3.5 The MENU Form

A menu is a screen that displays a list of possible options and provides a choice field where you enter a selection. You may stack menu selections, with certain restrictions. You may also select options that are not listed, when those options are valid within the framework illustrated in Figure 3-1.

Menus are the most powerful of forms. The flow of processing within ALL-IN-1 is largely controlled by:

- The presentation of menu options
- The Named Data associated with those options
- The search strategy used to match the two

An ALL-IN-1 subsystem is actually a tree of forms with a menu at the top. The form qualifiers, field qualifiers, and other Named Data entries of that top-level menu control the depth and nature of the subsystem's functionality.

Standard menu requirements are:

- The `.TYPE MENU` directive must be present when the form is called. This directive instructs the Form Processing facility to process this form as a standard menu.
- All menus must contain an enterable (FMS non display-only) choice field that is identified by the form qualifier `/CHOICE`. The choice field is managed by Form Processing as the area where the user always enters a menu selection.

3.5.1 Menu Named Data

Named Data in menus is used to store:

- The .TYPE MENU directive, the required /CHOICE qualifier, and any other optional form qualifiers
- The 1- to 31-character codes listed on the screen as options and the associated processing
- Hidden menu options and the associated processing
- Keyboard definitions and the associated processing
- The names of any fields, including the choice field, where user input can be solicited or data stored, and the associated field qualifiers

3.5.2 Menu Form Qualifiers

The only menu qualifier that is always required is the /CHOICE qualifier. The syntax of /CHOICE is:

/CHOICE = field

/CHOICE identifies the field in which the user enters a menu selection. The single, required argument is the field that corresponds to the FMS name of the enterable choice field.

For example:

Named Data

1 Name .TYPE
MENU/CHOICE = CFIELD/CLEAR/TITLE = TFIELD

The choice field can take all field qualifiers. The processing of the choice field contents is illustrated in the following section.

The common form qualifiers can be used optionally with menus. One common qualifier, /FIELDS, is processed slightly differently. The /CAPTIVE, /MORE, and /ONCE qualifiers are also valid with menus.

/CAPTIVE

The **/CAPTIVE** menu qualifier limits the menu-option search to the current form's or form set's Named Data and to any Named Data linked to the current form with **/MOREs**. **ALL-IN-1** usually allows the user much flexibility in navigating through menus. The Named Data of the default menu (usually form **DEFAULT**) is automatically searched for a match. If none is found in the Named Data of the current form or **/MORE-linked** forms and if the search of the default menu does not find a match, the option is treated as a form name and that form is displayed, if it exists and is available. **/CAPTIVE** should be used when it is necessary to restrict menu options to explicitly defined choices.

For example:

Named Data

1 Name .TYPE

MENU/CHOICE = CFIELD/CLEAR/CAPTIVE/OVERLAY/MORE = GFRM,HFRM

/FIELDS = field[,field...]

The **/FIELDS** form qualifier takes as arguments the names of one or more enterable fields that are to be activated while the menu is displayed. These fields are the only fields that participate; any other enterable fields on the form are ignored.

All form and field qualifiers associated with the **/FIELDS-activated** fields are processed typically. The current choice field must be among those explicitly activated by **/FIELDS** if the form is to be processed properly as a menu.

Because the menu choice field is treated specially, the **/FIELDS** qualifier has a different effect than in other form types. Generally, the choice field participates as the only enterable field, even when other enterable fields exist on the form. However, when **/FIELDS** activates additional fields, the form is processed as a menu/argument form hybrid. That is, the form's FMS-specified field-access order is followed, and the user navigates as in an argument form. The choice field participates in its proper order, but the input in the choice field still controls the flow of processing.

The **CHOICE** field contents are processed after **RETURN** is pressed and normal field processing completes.

Assume form EXAMPL contains the following Named Data entries:

Named Data

1 Name .TYPE

MENU/CHOICE = CFIELD/CLEAR/DATE = DATE/USER = USER

2 Name .TYPE

/GET = DOCNAM,\$EDITDOC

EXAMPL could be redefined from its own choice field (CFIELD) to simulate an argument form:

<FORM . /RESET/CHOICE = CFIELD/FIELDS = HIDDEN1,HIDDEN2,CFIELD/HI

In this function call, EXAMPL is still processed as a menu, but the /RESET qualifier clears out the existing form qualifiers. The choice field, CFIELD, must be redeclared and specified among the participating fields. The user enters fields HIDDEN1, HIDDEN2, and CFIELD in the FMS-defined order of access, and these are highlighted; once the form is filled and the user presses RETURN, the result depends on what has been entered in the CHOICE field.

/FIELDS is usually used in FORM function calls rather than in Named Data.

NOTE: *If the /BEGIN common form qualifier is present, the beginning field it identifies must be activated by /FIELDS.*

/MORE = form-symbol

OR

/MORE = "form-symbol-list"

The /MORE menu qualifier takes as arguments one or more ALL-IN-1 symbols that represent form names. ALL-IN-1 takes the Named Data sections of the specified forms and appends it, in order, to that of the qualified form. The resulting expanded Named Data section is then searched whenever the user enters something in the choice field of the qualified form.

The forms specified by `/MORE` do not have to be displayable forms. `ALL-IN-1` treats them simply as storage locations for Named Data information. However, they can be standard, fully independent forms. When a standard form is specified by `/MORE`, that form's `.TYPE` directive is ignored, and its other Named Data entries are processed as if they were part of the qualified form's Named Data. Thus, argument forms, entry forms, search forms, and so on, may be specified by a `/MORE` on a menu.

When `/MORE` is specified in a `FORM` function call for a form already containing a `/MORE`, the previous form names are discarded and the extended Named Data section is rebuilt using the new forms.

`/ONCE`

The `/ONCE` menu qualifier causes a menu to exit after a single successful selection is made in the choice field. If a non-existent menu option is entered, this invalid choice does not count as a successful selection. If an `ALL-IN-1` function or a `SCRIPT` command is entered, the choice does not count either.

Once a successful choice has been made on a menu qualified by `/ONCE`, then the next time that menu is reached (either by `EXIT SCREEN` back to the menu, or by going explicitly forward to it from another form), an automatic and immediate `EXIT SCREEN` is performed on the `/ONCE` menu.

Also, `/ONCE` can be used on overlay menus to allow the overlay to be treated as a one-time menu.

3.5.3 Processing Menus

When the Form Processing facility encounters the `.TYPE MENU` directive in Named Data, the form is preprocessed as follows:

- 1 All form qualifiers are preprocessed or flagged for postprocessing:
 - The field associated with the `/CHOICE` qualifier is recognized as the choice field.
 - If the `/FIELDS` qualifier is also present, the specified fields are activated; otherwise only the choice field is activated.

- If the /MORE qualifier is present, the Named Data of the specified forms is treated as an extension of the current form's Named Data menu options.
 - If the /PRE_FUNCTION qualifier is present, the function or functions it specifies are executed.
 - If the /DATE, /GET, /TITLE, or /USER qualifiers are present and active, the fields associated with them are filled with the specified values.
 - If any other common form qualifiers such as /CLEAR, /HELP, or /LOG are present, they are processed.
- 2 There may be other enterable fields, as designated by the /FIELDS qualifier, but the user is eventually placed in the choice field. Any choice field input that is not terminated by EXIT SCREEN (or some other key defined as EXIT SCREEN) is saved and parsed.

NOTE: *Since Field Processing is used to get the menu choice, all of the field processing capabilities may be used on the choice field. Thus, if the choice field contains, for example, a /VALID qualifier, then the choice is restricted to those options that match the criteria of the validation expression.*

ALL-IN-1

Professional Workstation

WP Word and Document Processing
EM Electronic Messages
DM Desk Management
TM Time Management
IM Information Management
BA Business Applications
COM Communications
M More Menu Options

TR Training
EX Exit Workstation

TIME

Enter selection and press RETURN

1 Name .TYPE
MENU /USER = USER/TITLE = TITLE/MAIL = MAIL/CHOICE = CHOICE/DATE = DATE

2 Name .TYPE
/MORE = 'MAIN2'/CLEAR/GET = DAY,OA\$DAY;MEETIN,

3 Name .TYPE
OA\$MEETING__COUNT__DISPLAY/HARD = "Main Menu"

4 Name TR
FORM CBIAGS

5 Name M
FORM MAIN2

6 Name .GOLD A
FORM MAIN2

7 Name CHOICE
/HARD = Options: WP,EM,DM,TM,IM,BA,COM,M,TR,EX

Figure 3-1 Menu Processing Flow Control

Assuming the Main Menu contains the Named Data entries shown in Figure 3-1, an option is processed as follows:

- 1 The form is preprocessed by the Form Processing facility. The /MORE points to another form, MAIN2, where the Named Data is continued. Form Processing treats the Named Data of forms linked by /MORE as a single, extended Named Data section if and when the search for a valid option leads beyond the current form.
- 2 The Input Processing facility determines the source of the input. If there is a response already stored from a previous prompt, such as when the user enters two or more stacked commands, the next response is acted upon. If an active script is providing input, then the next line in the script file is automatically injected into the choice field. Otherwise, the menu is displayed, if it is not already, and the user is prompted for a response.
- 3 The input is parsed in terms of an entry followed by a terminator. Valid entry/terminator combinations include:
 - <EXIT SCREEN>. The entry is ignored, and the prior menu displays.
 - Any listed or hidden menu option, followed by <RETURN>. These options can be identified:
 - In the Named Data of the displayed menu
 - In the Named Data of any forms specified with the .MORE directive
 - In the Named Data of any /MORE-linked forms
 - In the Named Data of the default form (usually form DEFAULT)
 - Any form name in any form library accessible to the user, followed by <RETURN>.
 - Any predefined or user-defined function-key sequence, such as Gold \$, Gold A, and so on. Usually the entry is ignored, but at least one predefined function key, Gold H (Help), acts upon the contents of the choice field; user-defined key sequences can also access and act upon the contents of the choice field. Predefined key sequences are maintained in the Named Data of form DEFAULT and may be redefined in the Named Data of any other form.

- Any interactive function call followed by `<RETURN>`. The syntax is `< command`. Calling functions interactively requires the PRVCMD User Profile privilege.
 - A dollar sign (\$), optionally followed by a DCL command, followed by a `<RETURN>`. Calling the DCL function interactively like this requires the PRVDCL User Profile privilege; see the DCL function.
- 4 ALL-IN-1 translates the parsed choice field entry into a function call or a series of function calls, and then invokes the call(s).

The menu processing loop continues until you call either the EXIT or BYE function.

3.5.3.1 Form DEFAULT. All searches for a match to a choice field entry on a non-/CAPTIVE menu automatically include as a last resort, the Named Data of form DEFAULT, which is a memory-resident form. DEFAULT is a blank (but displayable) menu containing Named Data entries that define frequently used menu options, terminators, and special-function keys. For example, the menu option EX is defined in DEFAULT: it is associated with the EXIT function, thus allowing you to exit ALL-IN-1 by entering EX from any menu.

ALL-IN-1 searches DEFAULT only after all other possible locations for a match have been searched. Any other form can be specified as the automatic default with the SET_MENU function.

The keypad definitions in DEFAULT (or any default menu) work for all forms, and menus, (even those with /CAPTIVE).

3.5.4 Stacking Menu Options

To reduce keystrokes and menu prompts, you can enter any succession of options on one or more menus as one stacked command. Each option is performed in turn, but some of the menus may not display. For example, the following sequence can be entered at any menu:

```
EM RN WP C<RET>
```

These four options are processed sequentially. The first thing the user sees is the next mail message, and when that has been read or exited from, the

WP Create Document argument form displays. When this form has been filled out or exited from, the WP menu displays. Another example follows:

FD SEL C FI<RET>

The Forms Development Select overlay displays first. After a form is selected, the Create File from Entry Form overlay displays. After this form exits, the Forms Development menu redisplay.

NOTE: *You can stack menu-choice options, but you cannot enter form names and interactive function calls in this manner. Also, if any option in this list displays a form with enterable fields, you must enter values for those fields directly: ALL-IN-1 does not interpret the next item in the choice field as input into a field.*

You can stack function calls interactively from a menu choice field using a different syntax. See also the Script facility, which provides tools for creating indirect command files.

3.5.5 Invoking Menu Options Indirectly

As covered earlier, Menu Processing looks to a menu's Named Data to determine how to process the contents of the choice field. With the OA\$MENU__REMAINDER special GET destination you can also indirectly simulate choice field entries. ALL-IN-1 translates the specified symbol__list and treats it as if you entered the same thing in the choice field and pressed RETURN when you call
GET OA\$MENU__REMAINDER=symbol__ list while the current form is a menu. This feature lets you associate commonly used menu options with key sequences. For example, the Named Data on form DEFAULT has the following entry:

```
1 Name .PF4  
GET OA$MENU__REMAINDER="SEL"
```

This allows the user to invoke the SEL option by pressing the PF4 key.

3.5.6 Creating and Processing Complex Menus

Menus are more versatile than other form types because of the variety of processing that can be directed through Named Data. They are also more versatile because of the variety of valid entries that can be made in the

choice field. The following subsections cover two ways to take advantage of the full range of menu processing.

3.5.6.1 Create Menus with Multiple Enterable Fields. As discussed under the `/FIELDS` menu form qualifier, when other enterable fields precede the choice field, those fields are ignored unless explicitly activated by `/FIELDS`. When other enterable fields follow the choice field, they may be entered by pressing `TAB` from the choice field.

When other enterable fields are activated by `/FIELDS`, the form is processed like an argument-form/menu combination. The user navigates through all `/FIELDS`-activated or enterable fields in the order specified when the form was created.

In this context, the `CHOICE` field is just another field, but it still remains the decision field on the form.

Fields other than the choice field can also be activated and entered singly through explicit `FIELD` function calls. When a single field is identified by a `FIELD` function call, it is processed independently.

A menu can also be processed as another form `.TYPE`. This approach is often more efficient than calling the `FIELD` function to activate one field at a time.

3.5.6.2 Menus with Additional Options on Other Screens. The `/More` form qualifier allows you to create either single- or multi-page menus that allow a user to enter one of a full range of options from any of several screens, regardless of whether the option is present on the currently displayed screen or in the associated Named Data.

Create a primary menu with the `/MORE` qualifier, and identify the additional options forms with the `/MORE`. If you wish these additional options forms to also be displayable menus, they should be standard menus with `.TYPE` entries, and should have the `/MORE` qualifier pointing to all the other linked menus. For example, form `FIRST` contains the following Named Data entries:

```
1 Name .TYPE  
MENU/CHOICE = CHOICE/MORE = SECOND,THIRD
```

```
2 Name A  
COMMAND A
```

```
3 Name M  
FORM SECOND
```

When this form displays, the user may enter options A, M, or any other valid option identified in forms SECOND or THIRD. The order in which forms are listed with the /MORE qualifier is important since this order determines how the expanded Named Data section is searched.

Form SECOND contains the following Named Data entries:

Named Data

1 Name .TYPE
MENU/CHOICE = CHOICE/MORE = THIRD,FIRST/OVERLAY = FIRST

2 Name B
COMMAND B

3 Name M
FORM THIRD

and form THIRD contains the following Named Data entries:

Named Data

1 Name .TYPE
MENU/CHOICE = CHOICE/MORE = FIRST,SECOND/OVERLAY = FIRST

2 Name C
COMMAND C

3 Name M
FORM FIRST Display No additional options available

Any option — A, B, or C — can be entered from any form — FIRST, SECOND, or THIRD. Note that all the forms contain the M option to display the next form in the chain. Every time the forms' Named Data is linked with /MORE, there are three M options, all with conflicting instructions. But because the current form's M always is encountered first, the option always works correctly. This is a general rule that applies to the /MORE qualifier.

Note that forms SECOND and THIRD are overlays. As menus, they are part of the prior menu stack and can be recalled by an EXIT SCREEN. As overlays, they only repaint part of the screen. Therefore, overlay menus that are intended as additional-topics screens should specify an underlying

form as the `/OVERLAY` argument. This prevents the overlay from being redisplayed with an `EXIT SCREEN` in a disembodied and potentially confusing manner.

3.5.7 Menu Stacks and Menu Levels

Summary of menu stacks and menu levels:

- Invoking a menu adds the menu to the menu stack, as long as the menu option is not already present on the stack. If the menu is found on the menu stack, the current menu stack is cleared of all subsequent menus.
- Levels of menu-stacks: a new menu level is created when:
 - The `OA$MENU__LEVEL__PUSH` function is invoked.
 - A `/CAPTIVE` menu is invoked and the `/CAPTIVE` menu is not found on the current stack.
 - A menu is invoked from a nonmenu.
- A menu-stack level is exited when:
 - An `UNWIND` function is performed from any form on that level.
 - An `EXIT SCREEN` is performed from the only remaining menu in that level, except for the first level.
 - `EXIT SCREEN` is performed and no non overlay forms remain in the current stack.
- Within a menu-stack level, menu-stacks of previous levels are invisible. For example, if you invoke menu `X`, where `X` is not on the current level's stack, this causes a new incarnation of `X`, even if `X` is on previous level(s). Stacked menu options from previous levels are also invisible.
- An operation (such as a menu-choice) which causes a new menu-level to be created, causes any stacked menu options from the (now) previous menu level to become invisible. When that menu-level is later returned to, the saved-away stacked menu options resume executing, at the menu option that was ignored when the new menu-level was invoked.

3.6 The Entry Form

The entry form, `.TYPE ENTRY`, is the primary ALL-IN-1 data-entry interface. Entry forms describe indexed Record Management Services (RMS) files and may be used to solicit additions to, updates of, inquiries into, and deletions from those files.

Standard entry forms have three requirements:

- The `.TYPE ENTRY` directive in Named Data.
- Either:
 - A `.FILE` directive and a corresponding Named Data entry that is the name of the file that the form will access. The primary key may optionally be specified in this entry.
 - OR
 - A `/DATA=` form form qualifier that has as its argument a form that contains the `.FILE` directive for the data file.
- A `/MODE=` mode form qualifier in Named Data to indicate how the form is to access the corresponding data file.

3.6.1 Entry Form Named Data

Entry form Named Data is used to store:

- The `.TYPE ENTRY` directive and either the `.FILE` directive or the `/DATA` form qualifier
- The `/MODE` form qualifier, and any other optional form qualifiers, all appended after the `.TYPE ENTRY`
- The names of any fields where user input can be solicited or data stored, and the field qualifiers associated with those fields
- Any keyboard definitions for the form, and the processing associated with those definitions

3.6.2 The Data Entry Process: Creating Files

Before entry forms can be used to store and retrieve data, the accompanying data file must first be created. The Forms Development menu offers a Create File option (C FI) that calls the CREATE function to create a file from a valid entry form. You can also call the CREATE function interactively.

CREATE uses the FMS field attributes and layouts to build the physical record layout, and it extracts any additional information that has optionally been provided in Named Data on how to index the file. Most Named Data information, such as /KEYS and other form qualifiers, is ignored during the file creation. The Named Data .FILE directive is the exception: it contains instructions on how to create the file. If .FILE identifies a File Definition Language (FDL) file specification, the FDL file is used only during CREATE. The following paragraphs detail the creation process.

To create an indexed file in which the first field will be the key, simply specify the file name in the .FILE directive. The syntax is:

Named Data

```
1 Name .FILE  
file-name
```

where file-name is the indexed file name. The first (FMS-specified) field on the form will be assigned as the key. After the file is created, it can be accessed through this form, through any form with the same field layout and the same .FILE directive, or through any form that references this form with a /DATA form qualifier.

For example, form INDEX could contain the following Named Data entries:

Named Data

```
1 Name .FILE  
ONEKEY.DAT
```

Specifying:

```
<CREATE INDEX
```

would create indexed file ONEKEY.DAT, indexed by whichever field is recognized by FMS as the first field on the form.

To specify some other FMS field as the primary key, specify that field explicitly in the .FILE directive. The syntax is:

Named Data

```
1 Name .FILE  
file-name,key
```

where key is any valid field on the form. For example:

Named Data

```
1 Name .FILE  
ONEKEY.DAT,PRIME
```

The FDL utility is covered in the VAX FMS documentation set. To use it in an entry form, specify:

Named Data

```
1 Name .FILE  
file-name,fdl-file-spec
```

where fdl-file-spec is the name of an FDL description file. The FDL file specification and an explicit primary key are mutually exclusive, and you must specify at least one key in the FDL specification.

As with other entry-form files, the FDL-created file may be accessed through the form it was created with, through any form with the same field layout and the same .FILE directive, or through any form that references this form with a /DATA form qualifier.

To create sequential files with an entry form, use the /MODE=ENTRY combination. You can also build sequential files using the MERGE function, search and select forms (explained in this chapter), the /LIST field qualifier and the NEXT_LIST function.

NOTE: When you call the CREATE function, ALL-IN-1 always treats the file's explicit or implicit primary key as an RMS key of reference. By default, the Key Entry facility indexes the file by this RMS key. However, when you access that file, you may specify any number of additional ALL-IN-1 keys with the /KEYS qualifier, described in a subsequent section.

ALL-IN-1 also supports FDL-created files with multiple RMS keys. These RMS keys are true alternate keys: they may be segmented and they may overlap. OAUSER.DOCDB.DAT is an FDL-created file that contains overlapping keys. The following section also illustrates a multikey, FDL-created file.

3.6.2.1 An FDL-Created File. Form AIE, the Action Item Entry form, provides access to the personal action items file, OAUSER:ACTITEM.DAT. Action Items, Reminders, and To-Do lists are stored in ACTITEM.DAT. To keep from confusing these, a segmented primary key is used to index the data file. Since there is no way to create an ALL-IN-1 entry form and specify that a group of contiguous or noncontiguous fields are actually one RMS key of reference, ACTITEM.DAT has been created with an FDL description file.

Assuming that form AIE has the Named Data entries in Table 3-1b, the following tables illustrate the fields and the .TYPE and .FILE directives with which the form has been created.

Table 3-1a Form AIE: Directives

Field Name	Attributes (key fields)	RMS Name
AI_TYPE*	Noecho, display__only	
AI_CLASS*	Uppercase, Underline	AI_ITEM_KEY
AI_ITEM*	Underline	
AI_SCHED_DATE_NBS*	Noecho, display__only	AI_SCHED_DATE_NBS
AI_SCHED_DATE	Underline	
AI_DESC1		
AI_DESC2		
AI_DUE_DATE		
AI_SCHED_DATE		
AI_PRIORITY*	Response__required, Underline	
AI_STATUS		
AI_CLOSE_DATE		
AI_CLOSE_DESC1		
AI_CLOSE_DESC2		

Key fields are starred (*).

Table 3-1b Form AIE: Field Information

Named Data
1 Name .TYPE ENTRY /MODE = UPDATE/KEY = AI__TYPE,AI__CLASS,AI__ITEM
2 Name .TYPE /POST__FUNCTION = "DATE__CONVERT AI__SCHED__DATE__NBS,AI__SCHED__DATE,7"
3 Name .FILE ACTITEM.DAT,OA\$LIB:ACTITEM.FDL
4 Name AI__CLASS /RSE__RECOG = AIE.AI__CLASS WITH .AI__TYPE EQS "A" AND .AI__CLASS = AI__CLASS
5 Name AI__CLASS /SHOW = '.AI__ITEM .AI__DESC1'/PUT__SAVE = \$AM__SELECT1
6 Name AI__ITEM /RSE__RECOG = AIE.AI__ITEM WITH .AI__TYPE EQS "A" AND .AI__CLASS EQS AI__CLASS
7 Name AI__ITEM AND .AI__ITEM = AI__ITEM /SHOW = '.AI__CLASS .AI__DESC1'/PUT__SAVE = \$AM__SELECT2
11 Name AI__STATUS /VALID = OA\$TABLE:"O,C"/HARD = 'Status'/GET__SAVE = "O"/BLANK
12 Name AI__PRIORITY /HARD = 'Priority'/GET__SAVE = "A1"/BLANK
13 Name AI__DUE__DATE /OPTIONAL/GET__SAVE = OA\$TM__DATE/BLANK/VALID = CAL\$CHECK__DATE/HARD = 'Due Date'
14 Name AI__CLOSE__DATE /OPTIONAL/VALID = CAL\$CHECK__DATE/HARD = 'Closure Date'
15 Name AI__TYPE /GET__SAVE = "A"
16 Name AI__SCHED__DATE /GET__SAVE = OA\$DATE__FULL/BLANK/VALID = CAL\$CHECK__DATE/OPTIONAL

Assuming form AIE contains the previous Named Data entries, Table 3-1 illustrates the following features:

- Fields `AI_TYPE` and `AI_SCHED_DATE_NBS`, with the FMS attributes of `noecho` and `display-only`, are hidden fields: they contain data but never display.

`AI_TYPE` is a segment of the primary key of reference (key 0), as defined in the FDL description file used to create `ACTITEM.DAT`. Named Data entry 15 illustrates the `/GET_SAVE` that always loads `AI_TYPE` with an A — for Action Item. Ticklers are always created and stored with `AI_TYPE` equal to T.
- The visible part of the primary key consists of two more concatenated fields: `AI_CLASS`, the user-supplied Action Item class; and `AI_ITEM`, the name of the individual item. The primary key thus consists of three fields, two of which are visible. This primary key has been given the RMS name `AI_ITEM_KEY`, as illustrated in Tables 3-1 and 3-2.
- To keep storage and retrieval orderly and simple, a secondary key has been created based on action-item date and priority. Again, this is a segmented key, and one of the participating fields, `AI_SCHED_DATE_NBS`, is hidden. A corresponding display field, `AI_SCHED_DATE`, contains the same date in a more meaningful format. The other participating field in the secondary key is `AI_PRIORITY`. This 2-character segment allows action items on the same date to be retrieved and displayed in order of importance. This two-part secondary key has been given the RMS name `AI_SCHED_DATE_NBS`, as illustrated in Tables 3-1 and 3-2.
- The `/KEY` qualifier on the `.TYPE` line indicates that once the file is created, three fields act collectively as the primary ALL-IN-1 key. These key fields correspond to the locations of the primary-key segments given in `OA$LIB:ACTITEM.FDL`.
- Table 3-1 shows that `OA$LIB:ACTITEM.FDL` and `ACTITEM.DAT` are linked through the `.FILE` entry. This FDL file is given as Table 3-2.

Table 3-2 File OA\$LIB:ACTITEM.FDL

TITLE		ACTION ITEM DATA BASE
IDENT	2-FEB-1984 13:29:58	VAX-11 FDL Editor
FILE	ALLOCATION NAME ORGANIZATION	1 ACTITEM.DAT indexed
RECORD	SIZE	369
KEY 0	CHANGES DUPLICATESno NAME SEG0__LENGTH SEG1__LENGTH SEG2__LENGTH SEG0__POSITION SEG1__POSITION SEG2__POSITION TYPE	no AI__ITEM__KEY 1 15 40 0 (FMS field AI__TYPE) 1 (FMS field AI__CLASS) 16 (FMS field AI__ITEM) string
KEY 1	CHANGES DUPLICATES NAME NULL__KEY SEG0__LENGTH SEG1__LENGTH SEG0__POSITION SEG1__POSITION	yes yes AI__SCHED__DATE yes 17 2 56 (FMS field AI__SCHED__DATE__NBS) 225 (FMS field AI__PRIORITY)
KEY 2	CHANGES DUPLICATES NAME NULL__KEY SEG0__LENGTH SEG0__POSITION	yes yes AI__SCHED__DATE__NBS yes 17 (FMS field AI__SCHED__DATE__NBS) 56

After you have created the FDL file, you must identify it in the Named Data of the entry form. The CREATE function call:

<CREATE AIE

then creates the file ACTITEM.DAT in the user's ALL-IN-1 directory.

3.6.3 The Data Entry Process: Accessing Data Files

.TYPE ENTRY identifies a form as an entry form. You must use one of the following methods to associate that entry form with the proper data file.

Either:

- The form can contain a .FILE directive in Named Data that identifies the data file and possibly a primary key.
- OR
- The form can contain a /DATA=form qualifier in Named Data that identifies another entry form, which in turn contains a .FILE directive.

The .FILE directive is independent of the .TYPE directive. A form can be created with neither, either, or both of these directives, increasing the form's flexibility.

Also, a form used as an entry form with the /DATA qualifier need not correspond exactly to the associated data file. ALL-IN-1 determines which fields match and processes the form accordingly.

Access to ALL-IN-1 data files is largely controlled by the /MODE=qualifier argument and by the key or keys specified.

3.6.4 Entry Form Qualifiers

The only entry form qualifier that is always required is the /MODE qualifier. Other optional qualifiers may also be specified on entry forms.

/MODE=mode

/MODE tells how the entry form is to be used on the file. The single,

required argument, mode, corresponds to the type of operation to be performed. The valid modes are:

ADD

Use the ADD Mode to add a record to the file. If the key to the record already exists, ALL-IN-1 signals an error.

Interaction between ADD and /VALID — When the mode is ADD and there is a /VALID field qualifier on the key field(s), adding records is impossible because the /VALID field qualifier overrides and contradicts the /MODE=ADD form qualifier. The /INVALID field qualifier is intended for use in conjunction with ADD Mode.

CHANGE

CHANGE Mode assumes that the user is updating an existing record and indicates an error if the record is not on file. The key fields may not be changed using this mode.

COPY

Use COPY to replicate a record in the file. Key fields may be changed using this mode.

DELETE

Use DELETE Mode to remove an existing record. ALL-IN-1 signals an error if the record is not on file.

ENTRY

Use ENTRY Mode to add a record to the end of a sequential file.

INQUIRE

Once the key or keys have been acquired, the INQUIRE Mode assumes that the user wants to view an existing record and indicates an error if the record is not on file.

UPDATE

UPDATE Mode prompts the user to choose from ADD, CHANGE, COPY, DELETE, and INQUIRE. The user decides to continue or exit after the transaction is completed. UPDATE is the most generalized key-entry mode.

If you create a form whose only enterable field is the key field, call it using:

<FORM formname/SAVE__START

Interaction between CHANGE, COPY, DELETE or INQUIRE and /INVALID — When the mode is CHANGE, COPY, DELETE, or INQUIRE and there is an /INVALID field qualifier present on the key field(s), accessing records is impossible because the /INVALID field qualifier overrides and contradicts the /MODE form qualifier. The /VALID field qualifier is intended for use in conjunction with CHANGE, COPY, DELETE, and INQUIRE Mode.

For example, the following four Named Data entries correspond to four separate forms, all of which describe the same data file:

Form ADD__TOPICS

Named Data

1 Name .TYPE
ENTRY/MODE=ADD

2 Name .FILE
TOPICS.DAT,TOPIC

...

Form UPDATE__TOPICS

Named Data

1 Name .TYPE
ENTRY/MODE=UPDATE/DATA=ADD__TOPICS

...

Form TOPICS

Named Data

1 Name .TYPE
ENTRY/MODE=INQUIRE/KEYS=FIELD A,FIELD B

...

Form DELETE__TOPICS*Named Data*

1 Name .TYPE
ENTRY/MODE = DELETE/KEYS = TOPIC,FIELDA

2 Name .FILE
TOPICS.DAT,TOPIC

...

A more efficient way to implement different modes on identical or similar forms is to override Named Data with a FORM function call.

Common form qualifiers can be used optionally with entry forms, with one exception: /CLEAR is invalid because it conflicts with entry-form preprocessing. Two other common qualifiers, /BEGIN and /FIELDS, are processed slightly differently by entry forms and are as follows. The /DATA, /ONE__ENTRY, and /KEY, /NOWRITE, /ONCE, and /SAVE__START entry-form qualifiers are also described.

/BEGIN = field

In entry forms, /BEGIN has as a single, required argument: the name of the field to enter after the key field(s) have been touched. The /BEGIN-specified field must be enterable.

The ALL-IN-1 Key Entry facility is independent of the Field Processing facility, and the /KEY qualifier (or the implicit key field) are not affected by /BEGIN. Thus, /BEGIN on entry forms identifies the first nonkey field.

/BEGIN does not restrict the user from either backspacing to a previous field or tabbing to subsequent fields.

Assume form LOGO has the following Named Data entry:

Named Data

1 Name .TYPE
ENTRY/MODE = UPDATE

Specifying:

<FORM LOGO /BEGIN = FLDB/KEY = FLDA,FLDC

would place the user in FLDA and FLDC first, in their FMS-specified order of access; followed by FLDB; followed by any other fields on the form.

Specifying:

<FORM LOGO /BEGIN = FLDC

would place the user in the default key field first, and then in FLDC (assuming that FLDC is not the key field, in which case the /BEGIN would be redundant).

/DATA = form

The /DATA entry-form qualifier has a single, required argument: the name of another entry form that contains the complete field definition of a data file and a .FILE directive naming the file. The .FILE directive is not required for the entry form that contains the /DATA qualifier and, in fact, is ignored if present. The /KEY qualifier is required instead to identify the primary key field or fields.

Forms that contain the /DATA qualifier need not have the exact same field layout as the file-description form. ALL-IN-1 determines which fields match and processes the form accordingly.

For example:

Named Data

1 Name .TYPE

ENTRY/MODE = UPDATE/DATA = PROFIL/KEY = USERNAME

...

This form updates the data file associated with form PROFIL, OA\$DATA:PROFILE.DAT, using USERNAME as the primary key.

NOTE: *The most widely used type of data set reference (DSR) is based on the entry form, and the form name is the data set name. However, you cannot build DSRs with entry forms that contain the /DATA qualifier. Specify instead the name of an entry form that contains a .FILE.*

`/FIELDS = field[,field...]`

In entry forms, `/FIELDS` takes as arguments the names of one or more enterable fields that are to be activated while the entry form displays.

By default, field processing processes all field qualifiers and solicits input from all enterable fields. The `/FIELDS` qualifier effectively redefines the form, creating a subset of fields that can be entered.

The ALL-IN-1 Key Entry facility is independent of the Field Processing facility, and the `/KEY` qualifier (or the implicit key field) is not affected by `/FIELDS`. Thus, ALL-IN-1 solicits input from the key fields regardless of whether they are specified by `/FIELDS`. However, once the user has supplied key input, they remain in a key field only if you have activated that field — either implicitly by default or explicitly by identifying the key fields with `/FIELDS`.

For example, calling entry form AIE without `/FIELDS` causes ALL-IN-1 to activate all the fields, and the Key Entry facility places the user in key field `AI_SCHED_DATE` after he or she enters valid key information. In contrast, calling:

```
<FORM AIE/FIELDS = AI_DESC1
```

causes the Key Entry facility to first solicit information for fields `AI_CLASS` and `AI_ITEM` and then to place the user in field `AI_DESC1`.

The order in which fields are touched after the key or keys have been supplied also depends on the presence of a `/BEGIN` form qualifier and the order of field access specified when the form was created.

NOTE: *If the `/BEGIN` form qualifier is present, that beginning field must be among those activated by `/FIELDS`.*

For example, assume the following Named Data entries were associated with form MYCAB:

Named Data

1 Name .TYPE

ENTRY/MODE = UPDATE/GET = EDITSTYLE,"PROFIL.EDITOR[\$USERNAME]"

2 Name .FILE

OAUSER:DATAFILE.DAT, DOCNAM

...

This form updates data file DATAFILE.DAT. The following FORM function call:

```
<FORM MYCAB/MODE = ADD/FIELDS = DOCNAM,SETUP/SAVE__START = OA$CURDOC
```

displays MYCAB, and stores the value of OA\$CURDOC in field DOCNAM. The user is in ADD Mode and has access to the field SETUP. The EDITSTYLE field is loaded by the /GET form qualifier during this call since /GET is processed before /FIELDS.

```
/KEY = field[,field...]
```

/KEY identifies one or more FMS fields to use as ALL-IN-1 keys when accessing a data file. Identifying a field as an ALL-IN-1 key automatically enables optional validation on that field. That is,

"/KEY=field__n" implies

"/VALID=field__n/OPTIONAL".

When you specify more than one field, they are ANDed together and processed collectively as the primary key. However, during the key-entry phase (that is, while you are entering values in the key field(s) and before you have pressed RETURN to select the record), pressing Gold L or Gold S in a key field displays the full range of entries possible for that current field. In other words, the key fields are not ANDed together until after you have entered all key fields and pressed RETURN.

To further refine recognition during the key-entry process with multiple keys, use the /RSE__RECOGNITION field qualifier on one or more key fields. This qualifier ANDs the key fields together when the user presses Gold L or Gold S in the qualified field.

Any participating, enterable field on the form can be designated a key field with /KEY. Key fields may be left blank, and any blank fields are ignored during the key-entry phase; but at least one key field must contain a value.

As discussed under the /DATA qualifier, /KEY is required when /DATA is present. In all other cases it is optional. By default, ALL-IN-1 assumes that the key or keys specified when the file was created with the CREATE function are the only keys used to access an indexed data file.

Since the ALL-IN-1 Key Entry facility is independent of the Field Processing facility, /FIELDS does not affect the key fields. The key(s) identified by the /KEY form qualifier do not have to be specified by the /FIELDS qualifier. Similarly, the /BEGIN qualifier does not affect the order in which the user enters /KEY fields. When /BEGIN and /KEY are both present, /KEY always determines the starting field(s) on the form.

The fields identified by /KEY are highlighted when the form displays. If you have created the fields with the typical background, the Key Entry facility highlights them in reverse video, and vice versa.

For example, assume the following Named Data entries were associated with form DOCDBQ:

Named Data

1 Name .TYPE
ENTRY/MODE = UPDATE

2 Name .FILE
DOCUMENT.DAT,DOCNAM

If DOCDBQ were called as is, the form would display in Update Mode, and the user would be required to enter something in field DOCNAM before proceeding. An alternate key can be specified as follows:

<FORM DOCDBQ/KEY = FOLDER

The mode is still UPDATE, but the key field is now FOLDER. To implement FOLDER as an additional key field rather than as a replacement, specify DOCNAM with the /KEY qualifier:

<FORM DOCDBQ/KEY = FOLDER,DOCNAM/BEGIN = TITLE

Input is solicited from FOLDER and DOCNAM; at least one of these fields must be filled. If field FOLDER (for example) were not qualified by any validation qualifiers in Named Data, then pressing Gold L or Gold S in this field displays all entries in the file that have a FOLDER value. Similarly, pressing Gold L or Gold S in DOCNAM displays all entries in the file that have a DOCNAM value, regardless of what has just been entered in field FOLDER. After a valid record is selected, the user enters field TITLE.

When the /SAVE__START form qualifier is used in conjunction with the /KEY, the order in which the fields are listed becomes important. The order in which /SAVE__START gives starting values must correspond to the order in which fields are specified by /KEYS. For example:

```
<FORM DOCDBQ/KEY = FOLDER,DOCNAM/SAVE__START = "$CURFDR,OA$CURDOC"
```

Field FOLDER is given the value of symbol \$CURFDR, and field DOCNAM is given the value of symbol OA\$CURDOC.

The /VALID, /INVALID, and /RECOGNITION field qualifiers can be used with alternate primary keys, as can data set references.

/NOWRITE

Generally, an entry form writes data to a file after successful completion of a transaction. The /NOWRITE qualifier overrides the write operation but otherwise does not affect the data-entry process.

For example:

Named Data

```
1 Name .TYPE  
ENTRY/MODE = ADD/DATA = DOCDB/NOWRITE
```

...

In this example, the user enters something in the key field or fields of the form, as determined when the data file associated with DOCDB was created. When sufficient key information has been given, the user fills in the remaining fields. When RETURN is pressed, the form and the fields are postprocessed, but the record is not written to the file.

/NOWRITE is usually dynamically appended to an entry form's Named Data. This qualifier allows you to take advantage of the side effects of an entry form — the field and form qualifiers — without altering the data file that the form describes.

/ONCE

When present, the /ONCE qualifier causes the entry form to process a single transaction only instead of cycling back to the key field after a transaction has been completed. If the transaction is valid, the form automatically exits without requiring user confirmation. See also the /ONE__ENTRY qualifier in the following paragraphs.

For example:

Named Data

```
1 Name .TYPE  
ENTRY/MODE = ADD/DATA = DOCDB/ONCE
```

...

This combination of entry form qualifiers allows the user to add one record; then the form exits.

/ONE__ENTRY

The */ONE__ENTRY* qualifier is similar to */ONCE*, but where */ONCE* causes a valid transaction to be automatically accepted, */ONE__ENTRY* displays prompts that allow a user to confirm a transaction.

For example:

Named Data

```
1 Name .TYPE  
ENTRY/MODE = ADD/DATA = DOCDB/ONE__ENTRY
```

...

This combination of entry form qualifiers allows the user to add one record; then the user is prompted to confirm the transaction.

NOTE: *The /SAVE__START form qualifier implies /ONE__ENTRY.*

/SAVE__START = symbol[,symbol...]

The */SAVE__START* entry-form qualifier gets a starting key value or values for a transaction, giving the user a starting key to work with. If more than one symbol is present, the string of symbols must be enclosed in quotes.

The argument(s) may be any ALL-IN-1 symbol: symbol-table symbol, data-file reference, or quoted string. The number of arguments specified depends on the number identified by the */KEY* qualifier. The order in which they are given depends on the order in which the corresponding fields are listed with */KEY*. When */KEY* is not present, only one */SAVE__START* symbol is required, and it is used to retrieve a record from the associated data file.

/SAVE__START implies the /ONE__ENTRY qualifier. That is, /SAVE__START allows one (confirmable) transaction only.

For example, assume that form EFORM has the following Named Data entries:

Named Data

1 Name .TYPE
ENTRY/MODE = ADD/SAVE__START = \$PHILO

2 Name .FILE
LOGOS.DAT,PKEY

In this example, the entry form displays with the record field PKEY filled with the value of permanent symbol \$PHILO. The remaining fields are also filled using the record associated with the key value. The user is placed in the first active key field.

<FORM EFORM/MODE = DELETE/KEY = ACCOUNT,IDCODE/SAVE__START = "\$ACCT,\$ID"

In this example, form EFORM has a key with two segments, corresponding to the FMS fields ACCOUNT and IDCODE. When the form is processed, these key fields are filled with the values of symbols \$ACCT and \$ID, respectively, and the designated record is retrieved before the form is displayed. ALL-IN-1 then prompts the user to confirm the deletion.

NOTE: *If the last character of a name is an E or a P (for example, SMITH P or SMITH E), then /SAVE__START=form qualifier inserts the mathematical value of P or E into the key field. To avoid this, modify the qualifier to have an extra set of quotes, for example:*

/SAVE__START = " 'SMITH E' "

3.6.5 Processing Entry Forms

The following occurs when an entry form is called:

- 1 If the /DATA form qualifier is present, then the specified file is opened. Otherwise, the file specified with the .FILE directive on the form is opened.
- 2 The key value must be determined in order to access the record to be updated, deleted, or added. If the /KEY form qualifier is present, the key field(s) identified by it are activated. If the qualifier is not present, the key field or fields assigned when the data file was created are activated. In either case, the user must supply a key value. The key fields are highlighted on the form.

If the /SAVE__START form qualifier is present, the value(s) specified are loaded directly into the key field(s), and user input to the key field(s) is bypassed.

Once the user enters valid key value(s) and presses RETURN, they can no longer modify those fields. The transaction continues with the next enterable field on the form.

3 The mode in which the file is accessed depends on the argument of the /MODE form qualifier:

- **UPDATE** — Read/write/update/delete/inquire, shared indexed file
- **ADD** — Write, shared indexed file
- **CHANGE** — Update, shared indexed file
- **COPY** — Copy, shared indexed file
- **DELETE** — Delete, shared indexed file
- **INQUIRE** — Read, shared indexed file
- **ENTRY** — Append information to a sequential file

4 Once the requested record is found, corresponding fields from the data file are stored on the screen. If the /STORE field qualifier is present on any field, the contents of the /STORE-specified field are retrieved from the file and placed in the qualified field.

Any field in the file that is not on the form is left unchanged (or left blank, if the mode is ADD) in the file. Any field on the form that is not in the file is not stored in the file (but the user may still enter data in the field).

5 If the mode was specified as UPDATE, then the user is prompted for the specific action to be performed (ADD, CHANGE, COPY, DELETE, INQUIRE). In all other cases, the action is taken from the /MODE qualifier.

6 The action is validated. The following are error conditions:

- Adding a record that is already on file
- Changing, copying, deleting, inquiring about a record that is not on file

- 7 If the record is being added, changed, copied, or deleted, field processing is called to get the rest of the fields on the form. If the /FIELDS form qualifier was specified, then those fields are activated. If it was not specified, then all fields are activated. If the /BEGIN qualifier is present, then field processing begins on that field. Otherwise, the first nonkey field is taken as the starting field. The user is not allowed to back up into the key fields unless the specified mode is COPY.
- 8 When the user presses RETURN, the corresponding fields from the form are stored into the file (/STORE may direct the form field to a different RMS field).
- 9 Before the transaction is completed, ALL-IN-1 prompts the user to confirm that the transaction is correct. This prompt is suppressed if the /ONCE form qualifier is present.
- 10 The record is then written (back) to the file. In the case of DELETE, the record is deleted, unless the /NOWRITE qualifier is present.
- 11 If the /ONCE, /ONE__ENTRY, or /SAVE__START form qualifiers are present, then the next/prior menu displays. Otherwise, the user is prompted for another key, and processing continues. An EXIT SCREEN from the key prompt causes the Key Entry facility to exit.

3.7 The Argument Form

Argument forms, .TYPE ARG, are functionally the simplest of ALL-IN-1 forms. They are primarily used to solicit information, or arguments, that can be used by ALL-IN-1 applications immediately or stored in symbols for later use.

Standard argument forms have a single requirement, the .TYPE ARG directive in Named Data.

3.7.1 Argument Form Named Data

Argument form Named Data is used to store:

- The required .TYPE ARG directive
- Any optional form qualifiers, all appended after the .TYPE ARG

- The names of any fields where user input can be solicited or data stored, and the associated field qualifiers
- Any keyboard definitions for the form, and the associated processing

3.7.2 Argument Form Qualifiers

All common form qualifiers can be used optionally with argument forms.

3.8 The Calc and Desk Forms

The calculation, or calc, form provides a means of performing simple, repeated operations in an application. A calc form typically solicits algebraic expressions, evaluates them, and displays the results or any other information as directed. The related Desk Calculator form provides a noncustomizable, electronic-calculator screen interface for evaluating algebraic expressions.

Calc forms are functionally similar to argument forms, but they behave like entry forms in that, after each calculation, the form remains on the screen and the user is placed in the first enterable field. The user must exit with an EXIT SCREEN. (Note: the user must exit from the Desk Calculator form from Gold Q instead of EXIT SCREEN.) To override this cycling feature, call a calc form as an argument form: `.TYPE ARG`.

The actual computations performed on a calc form are implemented through field qualifiers, typically `/COMPUTE`, `/DECIMAL`, `/GET_SAVE`, and `/PUT_SAVE`.

Standard calc forms have a single requirement, the `.TYPE CALC` directive in Named Data.

3.8.1 Calc Form Named Data

Calc form Named Data is used to store:

- The required `.TYPE CALC` directive
- Any optional form qualifiers, all appended after the `.TYPE CALC`

- The names of any fields where user input can be solicited or data stored, and the associated field qualifiers
- Any keyboard definitions for the form, and the associated processing

3.8.2 Calc Form Qualifiers

All common form qualifiers can be used optionally with calc forms.

3.8.3 The Desk Calculator Form

The Desk Calculator form is a specialized screen that provides a running-total capability and three permanent memory registers, in addition to the features of the COMPUTE function and the /COMPUTE field qualifier. The Desk Calculator form performs like a simple electronic calculator.

Enter numbers as you would on a regular calculator. Separate the numbers with any of the usual function keys. Chained operations are also allowed. For example, to obtain a running total while evaluating the expression:

$15.3 * 13.7 / 4 + 19 - 11$,

do the following:

Enter	Then Press	Display
15.3	* (mult)	15.3
13.7	/ (divide)	209.61
4	+ (add)	52.4025
19	- (subtract)	71.4025
11	=	60.4025

You can use the following arithmetic operators:

- + Add
- Subtract
- * Multiply
- / Divide
- ^ Exponentiation
- = Equal (Total)

3.8.3.1 Formula Mode. To switch from Desk Calculator Running-Total Mode to standard calc-form Formula Mode, press F, and then type in the formula. End the formula by pressing the TAB key, if you want to continue in Formula Mode, or the RETURN key to return to the normal Running-Total Mode. For example:

<F>(3 + 4) * (-1/.532) + 3.1³<TAB>

3.8.3.2 Changing the Number of Decimal Places. To change the number of decimal places displayed, enter P and then the number of decimal places you want displayed (up to 9). If you enter F, 15 decimal places are displayed. Scientific (floating-point) notation is used when the result exceeds the size of the display field.

3.8.3.3 Calculator Functions. You can use the following functions while in Desk Calculator Mode:

- A Absolute Value
- N Nearest Integer
- R Square Root
- C Clear Calculator (except memory)
- E Clear Current Entry

3.8.3.4 Using Memory Registers. While in Formula Mode, you can refer to the memory registers (A, B, and C) in your expression as you would refer to symbols in a COMPUTE-function expression. For example, to store a value in register A:

A = B + 5.6

5.6 is added to the contents of register B and stored in register A. You may also use the display register (X) in expressions.

Memory registers are saved permanently in the permanent symbol table, OAUSER:username.PST. If you leave ALL-IN-1 and then return, the values in registers A, B, and C are restored. A memory register can be cleared by entering Formula Mode and setting the value of the register to 0; for example:

A = B = C = 0

3.9 The Search Form

Search forms provide an interface that allows collection, retrieval, and presentation of data. The Search form, `.TYPE SEARC`, solicits record selection criteria from the user and usually presents a record selection expression (RSE) to the ALL-IN-1 Search facility. The RSE, specified in Named Data, comprises a data set to search and a set of instructions on how to match fields on the search form against fields in the data set.

Search forms are similar to select forms, but search forms offer additional file-searching capabilities and provide a direct way to save the record collection in a file.

Search forms have a single requirement, the `.TYPE SEARC` directive in Named Data.

Named Data

1 Name `.TYPE`
`SEARC [VMS-file], DATA SET, BOOLEAN EXPRESSION`

where

VMS-file is an optional argument that names a sequential file to contain the keys of the records that match the RSE or DSR. This file is referred to as a selection list, and the default file-type extension is `.SEL`. If the VMS file specification is not present, no selection list is output.

Other forms and applications can access this selection list through the `/LIST` field qualifier and either the `NEXT__LIST` key or the `NEXT__LIST` and `OA$FLD__NEXT__LIST` functions.

Usually the data set is the name of an entry form that describes an indexed data file to search. However, you can search open-ended data sets that are not defined by a single file: the entire File Cabinet, for example, is a data set. The data-set syntax for accessing file-cabinet information is different than that for single entry-form related data files.

A Boolean expression is a set of instructions for searching the data set. The Boolean expression is usually required, but some special data sets, such as `OA$DIR`, do not require additional search instructions. Boolean expressions are covered under the `FOR` function.

As mentioned, only one data set may be specified in a search-form RSE. Another difference between search-form RSEs and those specified elsewhere is the syntax: a comma separates the data set from the Boolean expression, not the optional keyword **WITH**.

3.9.1 Search Form Named Data

The Search form Named Data is used to store:

- The required **.TYPE SEARC** directive
- Any optional form qualifiers, all appended after the **.TYPE SEARC**
- The optional **.SFILE** directive which provides instructions on searching a file or files for text strings that are entered on the search form
- The optional **.SOUT** directive, which provides screen formatting instructions for record collections that are to display on the terminal
- The names of any fields where user input can be solicited or data stored, and the associated field qualifiers
- Any keyboard definitions for the form, and the associated processing.

Assume the Word Processing File Cabinet search form, **WPSEARCH**, contains the Named Data entries shown in Figure 3-2.

Document Index

Folder: _____

Title: _____

Number: From: _____ To: _____

Keywords: _____

Enter the fields you want to include in the search or press RETURN to search this folder, or press EXIT SCREEN for menu

```
1 Name .TYPE
SEARC WPLIST.SEL, CAB$, .FOLDER < = > FOLDER AND .TITLE < = > TITLE

2 Name .TYPE
AND (.DOCNUM EN DOCNUM1 OR .DOCNUM EN DOCNUM2 OR ( .DOCNUM GE

3 Name .TYPE
DOCNUM1 AND .DOCNUM LE DOCNUM2 ) ) AND .KEYWORDS < = > KEYWORDS

4 Name .TYPE
/OVERLAY

5 Name .SOUT
FOLDER:1:30:FOLDER,DOCNUM:32:6:NUMBER,TITLE:39:72:TITLE
.
.
.
```

Figure 3-2 Form WPSEARCH and Assumed Named Data

3.9.2 Search Expressions in the .TYPE Directive

In Figure 3-2, the Boolean expression provides the criteria by which the data set is searched and thus controls the behavior of the search form. DSRs are generally used for more restricted searches.

Assuming that form WPSEARCH contains the previous Named Data entries, the RSE is explained next.

Named Data

1 Name .TYPE

SEARC WPLIST.SEL, CAB\$, .FOLDER < = > FOLDER AND .TITLE < = > TITLE

2 Name .TYPE

AND (.DOCNUM EN DOCNUM1 OR .DOCNUM EN DOCNUM2 OR (.DOCNUM GE DOCNUM1

3 Name .TYPE

AND .DOCNUM LE DOCNUM2)) AND .KEYWORDS < = > KEYWORDS

The first parameter in the .TYPE directive is the form .TYPE, SEARC. The next parameter is the selection list, WPLIST.SEL, which will be created to store the key values of the matching records. The remainder of the .TYPE directive is the RSE.

CAB\$ is the generalized syntax for accessing the File-Cabinet data set.

The Boolean expression in Figure 3-2 begins with .FOLDER. Reading from left to right, the entire .TYPE directive translates to:

Find all File-Cabinet documents:

- Whose FOLDER names contain the FOLDER names entered on WPSEARCH.
AND
- Whose TITLES contain the TITLES entered on WPSEARCH.
AND
- Whose DOCNUMs (system-assigned document numbers) fall within the range of DOCNUM1 and DOCNUM2 on WPSEARCH.
AND

- Whose KEYWORDS contain the KEYWORDS entered on WPSEARCH
- If fields DOCNUM1 and DOCNUM2 are left blank, treat the blanks as match-all entries.

The compound test on DOCNUM1 and DOCNUM2 illustrates a specialized RSE operator that is especially useful in search forms. This operator, EN, stands for equal or null and equates to true when the value being compared against — in this case, DOCNUM1 or DOCNUM2 — is blank or zero. That is, if the user leaves all fields blank, then all records are selected since they all match the equal or null condition on DOCNUM1 and DOCNUM2.

Similarly, the <=>, or CONTAINING, operator, equate to true when the value being compared against — in this case, FOLDER or TITLE — is blank. Thus, pressing RETURN on WPSEARCH when all the fields are empty results in the entire File Cabinet being collected.

Other examples follow.

Named Data

```
1 Name .TYPE  
SEARC SRCHLST.SEL, CAB$, .DOCNAM = = DOCNAM OR .TITLE EQUAL TITLE
```

Again, this search form searches the File-Cabinet data set. Any matching records output to file SEARCHLST.SEL. The search process for this example is as follows:

- 1 This form displays. The user enters information in fields DOCNAM and/or TITLE.
- 2 When the user presses RETURN, the Search facility uses the RSE to determine how to search the File Cabinet. In this case, all file-cabinet records with field .DOCNAM exactly matching the contents of form WPSEARCH field DOCNAM are selected, as are all records in which field .TITLE matches the contents of form WPSEARCH field TITLE. If the RSE had been:

```
DOCNAM = = DOCNAM AND .TITLE EQUAL TITLE
```

OR

```
.DOCNAM EQS DOCNAM AND .TITLE = = TITLE
```

or any combination of operators that means equal to string, only those records in which both fields matched exactly are selected. Leaving fields DOCNAM and TITLE blank does not result in a match-all condition; this only occurs with the EN operator.

Named Data

```
1 Name .TYPE  
SEARC MS.SEL, CAB$, .DOCNAM < = > DOCNAM AND .HIDDEN EQS "A"
```

In this example, all records in which the RMS field .DOCNAM contains the value of FMS field DOCNAM and the RMS field .HIDDEN contains an A, are collected.

- 3 If an error occurs in the parsing of the .TYPE line, a message is generated in which a caret (^) points to the location of the error.

Search forms select records similarly to FOR-function calls and select forms; the main difference is how the selected records are acted upon. While the action in search forms is implied by the selection-list parameter and by the .SOUT and .SFILE directives, the action in select forms and FOR-function calls is identified by special DO subfunctions. Select forms also have optional form qualifiers.

3.9.3 Search Form Qualifiers

The common form qualifiers can be used optionally with search forms.

3.9.4 Searching for Strings: The .SFILE Directive

You can also use the search form to search sequential files for specified text strings. First, define fields in the search form that contain text strings for which to search. The .SFILE Named Data entry then identifies the field in the data file that contains the VMS file name of the text file. The

specified file is opened and the program scans the file for the occurrence of any of the user-specified text strings. The format of the .SFILE entry is:

Named Data

1 Name .SFILE
file__name__field,bool__operator,text__search__string(s)

where

file__name__field is the field in the entry form identified in the form .TYPE directive which contains the VMS file name of the text file to search.

bool__operator is either the Boolean operator AND or OR. This operator specifies how the search fields are to be combined; that is, the search strings are all ANDed together or ORed together.

text__search__string(s) are one or more field names from the search form that contain the text strings to search. Any field or form qualifiers on these fields are processed normally.

For example:

Named Data

1 Name .SFILE
FILNAM,OR,SEA1,SEA2,SEA3,SEA4,SEA5

If fields SEA1, SEA2, SEA3, SEA4, and SEA5 all contain phrases, then every designated file that contains at least one of these phrases is collected. The designated file is a VMS file identified in field FILNAM; as mentioned before, this file is actually two levels removed from the search form. FILNAM is a field on the entry form associated with the file and not a field on the search form. Also FILNAM just contains the name of a VMS file to be searched, nothing more.

3.9.5 Formatting Record Collections: The .SOUT Directive

The .SOUT formats a collection of selected records and displays the collection in a scrolled region on form SEARCHKEY or on form SEARCH132. A header displays in a title field on the output form.

The format of the .SOUT entry is:

Named Data

1 Name .SOUT
field:start:length:header[,field:start:length:header. . .]

where

field is the name of a field in the data file to output.

start is the zero-origin starting column on the screen where the field displays.

length is the number of columns the field occupies.

header is a heading to print at the top of the screen for this field.

These four parameters may be repeated as necessary. For example:

Named Data

5 Name .SOUT

FOLDER:1:30:FOLDER,DOCNUM:32:6:NUMBER,TITLE:39:72:TITLE

For each matching record in the searched file, the contents of fields FOLDER, DOCNUM, and TITLE display. FOLDER fills columns 1 through 30 under FOLDER. DOCNUM fills columns 31 through 36 under the header NUMBER. TITLE fills columns 38 through 104 under the header TITLE.

3.9.6 Search Form Processing

Search forms are processed as follows:

- 1 The form displays.
- 2 The user enters values in as many enterable fields as desired for a match. Leaving a field blank creates an always-match condition when the CONTAINING or EN operators are specified; other operators interpret the contents of the field literally. If the form only contains display-only fields, the Search facility acts upon the contents of the fields after form preprocessing.
- 3 The Search facility compares the contents of the search-form fields against fields in the data set, as guided by the Boolean expression.
- 4 If all fields for a record meet the match criteria, the search form checks for a .SFILE entry.
- 5 If a .SFILE entry is found, the Search facility opens the text file and scans for the specified text string.
- 6 If all match criteria for data fields and text strings are satisfied, the Search facility then checks to see if a selection-list file is present. If present, the record's key is output to the VMS file name specified in the .SEARC directive.
- 7 The .SOUT directive, if present, causes Search to format the selected record on the user's terminal.
- 8 The cycle repeats until all records in the data set are scanned.
- 9 At the end of the search, the number of records selected displays. If any searching for text strings was performed, the number of lines scanned also displays.

3.10 The Select Form

The select form displays one or more records from a collection extracted from one or more data sets. The select form selects and retrieves data in much the same way as the search form, but in addition to displaying information, the select form also allows the user to select one record from the collection.

The requirements for standard select forms are:

- The `.TYPE SELECT` directive in Named Data. The `.TYPE` directive always specifies a `FOR` function call, with its record selection expression (RSE) and associated `DO` subfunction.
- A display-only scrolled region named `DISPLAY`. Scrolled regions are multiline fields identified by a single name. They are built with the FMS forms editor when the form is created. Field `DISPLAY` is used to display the record collection assembled by the `FOR` function.

3.10.1 Select Form Named Data

Select form Named Data is used to store:

- The required `.TYPE SELECT` directive and its associated `FOR` function call
- Any optional form qualifiers, appended after the entire `FOR` expression
- The names of any fields where user input can be solicited or data stored, and the associated field qualifiers
- Any keyboard definitions for the form, and the associated processing.

3.10.2 Select Form Qualifiers

You can add the common form qualifiers to select forms. You can also specify the `/LIST` and `/STYLE` select-form qualifiers. Use these optional qualifiers to:

- Control the output of the `FOR`-created record collection.
- Redirect the action of the `DO` function without re-entering the entire `.TYPE` line in a `FORM`-function call.

`/LIST` = output-list-symbol

The `/LIST` qualifier creates a sequential file with selected records. The required output-list-symbol argument is an `ALL-IN-1` symbol representing the VMS file name of the list.

`/LIST`, by default, serves the same purpose as the selection-list parameter on search forms. In both cases, a sequential list is created with just the key values of each record. You then use this selection list with the `/LIST` field qualifier and with the `NEXT__LIST` function. (By convention, selection lists have a default file-type extension of `.SEL`.)

However, you can also use `/LIST`, in conjunction with `/STYLE=FILE`, to create a listing of entire records. See also `/STYLE` in this section.

NOTE: *The `/LIST` select-form qualifier is distinct from the `/LIST` field qualifier. The qualifier described here creates a file; the `/LIST` field qualifier identifies an existing selection list (that may have been built with the `/LIST` select-form qualifier).*

For example:

Named Data

```
1 Name .TYPE
SELECT FOR EFORM1,EFORM2 WITH .EXAMPLE EQS "TEST"
```

```
2 Name .TYPE
DO PROMPT "Field EXAMPLE contains " TEST/LIST = "TEST.SEL"
```

NOTE: *`/LIST` is only valid with the `SEL__STYLE` subfunction.*

`/STYLE = keyword`

The `/STYLE` qualifier takes a single keyword argument that determines how the select form outputs the record collection. You can specify one of the following keywords:

- **CHOICE** — `/STYLE=CHOICE` displays the record collection and offers the option of selecting one record.
- **DISPLAY** — `/STYLE=DISPLAY`, the default, causes the select form to output the record collection through scrolled field `DISPLAY` on the current form.
- **NODISPLAY** — `/STYLE=NODISPLAY` suppresses record collection output. Use this qualifier in conjunction with `/LIST` when all you want to do is save the record collection keywords in a selection list.

- **FILE** — Like `/STYLE=NODISPLAY`, `/STYLE=FILE` suppresses record collection output and is intended for use with `/LIST`. However, `/STYLE=FILE` loads the specified list file with entire formatted records, not just key values.

In all other cases, `/LIST`, when present, creates a selection list based on keywords of selected records.

- **SELECT** — Like choice, **SELECT** displays the record collection, but allows multiple items to be chosen.

NOTE: `/STYLE` is intended for use with the `SEL_STYLE` subfunction. `SEL_STYLE` formats the record collection, and `/STYLE` determines how to output the formatted records.

3.10.3 Selecting and Processing Records

Assume form WPINDX contains the following Named Data entries. The `.TYPE` data entry is described next.

```
1 Name .TYPE
SELECT FOR CAB$ WITH .FOLDER = FOLDER AND .TITLE < = > TITLE
```

```
2 Name .TYPE
AND .DOCNUM EN DOCNUM1
```

```
3 Name .TYPE
AND .KEYWORDS < = > KEYWORDS
```

```
4 Name .TYPE
DO SEL_STYLE .FOLDER:20 " " .DOCNUM:7:Z .TITLE:40
```

```
5 Name .TYPE
/OVERLAY/GET = INDEX,#INDEX/STYLE = DISPLAY/LIST = "WPLIST.SEL"
```

SELECT FOR and **DO** are keywords required either by the select form `.TYPE` or by the **FOR** function. The form `.TYPE`, **SELECT**, and the `/OVERLAY` qualifier on line 5 are standard Named Data entries. The **FOR** function call encompasses an **RSE**, a **DO** subfunction, and the parameters of that subfunction, as described in the following list.

- 1 The **RSE** first identifies the data set (the ALL-IN-1 File Cabinet) through the expression `CAB$`. All subsequent references to fields are to fields in a File Cabinet related file (usually `OAUSER:DOCDB.DAT`).

- 2 The rest of the RSE matches several fields on form WPINDEX to the corresponding fields in the data set.
- 3 Records that meet the search criteria are selected and acted upon, one at a time, by the DO function SEL_STYLE. SEL_STYLE is a special FOR subfunction that formats records for output.
- 4 For each matching record, the first 20 characters in the folder name are displayed, followed by a blank, followed by the first 7 (zero-suppressed) digits in the document number, followed by the first 40 characters in the title.

SEL_STYLE is one of three special FOR subfunctions. Any valid ALL-IN-1 function may be called in its place. The three special subfunctions are:

- SEL_DISPLAY — Displays the record in the format specified.
- SEL_CHOICE — Displays the records and offers the option of selecting one record.
- SEL_STYLE — Formats records, and dynamically selects either SEL_DISPLAY or SEL_CHOICE based on the keyword given with the /STYLE qualifier.

In this case, the keyword on the /STYLE qualifier is DISPLAY, and the selected records are displayed.

The /LIST qualifier directs the output to file WPLIST.SEL.

3.10.3.1 Form WPINDEX Flow Control. Form WPINDEX is first processed as an argument form. In this example there are five enterable fields. Next, the FOR-function call is extracted from the .TYPE line and passed to the FOR function. The DO function acts upon a matching record or records. In this case, SEL_STYLE is called, and since the /STYLE qualifier is also present and identifies the style as DISPLAY, SEL_STYLE takes the field names and formatting instructions given to it and formats the print line for display.

The formatted print line is then output to the scrolled field called DISPLAY on the current select form. When field DISPLAY is full, ALL-IN-1 prompts for a RETURN to continue or an EXIT SCREEN to exit.

If the /LIST qualifier is present, the specified file is created and loaded with the keys of the selected records.

If the `SEL__CHOICE` function is called instead of the `SEL__STYLE` function, or if the `CHOICE` style is indicated, then a line number also outputs on the form, and another prompt allows the user to select one of the displayed records.

Any valid `ALL-IN-1` function or list of functions may be specified as the `DO` action function. When more than one function is called, separate the individual functions with two backslashes. For example, assume form `FORMCHANGE` contained the following Named Data entries:

Named Data

1 Name .TYPE
SELECT FOR PROFIL WITH .DEPART EQUAL DEPART

2 Name .TYPE
DO WRITE CHANGE PROFIL USER = .USER, START = .START FORM \ \ PROMPT OA\$SEL__KEY

3 Name .TYPE
"*s starting form is now* " START; press RETURN"

This example first displays `FORMCHANGE`. Presumably a department name and a starting form name are entered in fields `DEPART` and `START`, respectively. Then, for each individual in the specified department, the select `FOR` function updates the `START` field in the user profile to the specified starting form. A prompt is displayed after each update, indicating whose profile record is updated.

The following special symbols contain information that is generated by the `FOR` function during the selection process. These symbols contain the same information regardless of how `FOR` is called.

- `OA$SEL__ADDR` — Relative File Address of last selected record (used when selected key values are nonunique). `OA$SEL__ADDR` may be used as a key value in any `DSR`. It is equivalent to `OA$SEL__KEY` but provides faster access to the data.
- `OA$SEL__COUNT` — Number of records selected by `FOR`. The value of this symbol is only meaningful after the selection process is over.
- `OA$SEL__DSAB` — Name of the data set (usually an entry form name) that the records were extracted from.

- **OA\$SEL__KEY** — Contains the key value of the record selected by the user.
- **OA\$SEL__LINE** — Contains the current line number while doing a select. **OA\$SEL__LINE** is normally used as one of the symbols in a **SEL__CHOICE** function.
- **OA\$SRC__COUNT** — The total number of records scanned.

To overview, the select form depends on other features to implement its tasks:

- The Field Processing facility. When the select form is called, it is first processed as if it were an argument form. The form may optionally contain enterable fields. If it does not, then, with the exception the side effects of field processing (**/GET__SAVE**, **/PUT__SAVE**, and so forth), control passes to the next step.

If there are enterable fields, then the user can fill them in. Fields entered by the user can then be used in the selection expression. The **EXIT SCREEN** key is recognized and allows the selection process to be halted.

- The data-set search is initiated using the **FOR** function. The parameters for the search are provided in the **.TYPE** entry in Named Data or are passed in the **FORM** function call.

This step actually opens the data set(s) specified in the **FOR** function **RSE**. For each record that matches the selection expression, one or more functions (as specified after **DO** in the **.TYPE** entry) are called.

- The records are processed as specified by these **DO** function(s). These specialized **FOR** subfunctions provide a variety of options:
 - **SEL__CHOICE**
 - **SEL__DISPLAY**
 - **SEL__STYLE**

3.11 EXIT SCREEN and the PRIOR Form

In the Named Data of form DEFAULT, keypad 0 is associated with function OA\$FLD_EXIT, the EXIT SCREEN function. When it is pressed, the following algorithm determines which form is displayed next.

- 1 If the /NEXT qualifier is present on a form, that particular form is displayed.
- 2 Otherwise, successively prior forms on the menu stack are scanned to find the latest NON-OVERLAY form. This form is displayed. For example, enter the following sequence to stack menu options: A B OV1 OV2 C <RET>. If OV1 and OV2 are overlay forms, an EXIT SCREEN from C exits to form B.

With the /NEXT form qualifier, the /PRE_FUNCTION and /POST_FUNCTION qualifiers, and keypad definitions, it is possible to make FORM function calls to any type of form while any other type of form is current. When a menu is called while a nonmenu is current, a new prior menu stack is started. That is, pressing EXIT SCREEN on a menu called from a nonmenu recalls the nonmenu (and not some prior menu).

EXIT SCREEN unwinds to previous menu-level(s), but it does not unwind beyond the first level. This would cause an exit out of the ALL-IN-1 image.

3.12 Dynamically Redefining Named Data

One of the advantages of separating a form from its function is the ability to reuse the form in a new context. That is, you can create a standard form as one form .TYPE and use it solely as that form type. But you can also use form and field function calls to redefine Named Data entries and cause the form to be processed with new information or in an entirely different manner.

For example, the form qualifiers on a menu can be reset to present new information in display fields and/or alter the way in which the menu processes options. The menu can also be processed as an argument form, either with the same form qualifiers and the same participating fields or with new qualifiers and fields.

Menus are a special case. Techniques for redefining the Named Data of other form types are covered in the following sections.

3.12.1 The Form .TYPE Parameter in FORM Function Calls

To override the form type, specify the new form type in a FORM function call. For example, assume the Named Data of form EXAMPL contains the following .TYPE entry:

Named Data

```
1 Name .TYPE
MENU/CHOICE = CHOICE/GET = TOPFIELD,$VALUE
```

The following interactive FORM function call overrides the form .TYPE and causes EXAMPL to be processed as an argument form:

```
<FORM EXAMPL ARG/FIELDS = TOPFIELD
```

When this function call is specified, only field TOPFIELD participates in the form. However, all other applicable form qualifiers specified for menu EXAMPL (in this case, just the /GET, since /CHOICE is meaningless in argument forms), are still recognized. The /RESET qualifier negates this feature, as discussed next.

The FORM function is not limited to interactive calls. In fact, a FORM function call that redefines a form can be specified from that form itself:

Named Data

```
1 Name .TYPE
MENU/CHOICE = CHOICE/GET = TOPFIELD,$VALUE
.
.
.
8 Name XYZ
FORM . ARG/GET = TIME," 'Current time = 'OA$TIME"/FIELDS = TOPFIELD,BOTMFLD
```

Menu option XYZ causes the currently displayed menu (as indicated by the substitution character ".") to be processed as an argument form. Only fields TOPFIELD and BOTMFLD participate, and the form is then processed as a two-field argument form. Any keyboard definitions in Named Data are still recognized with the new form type.

When users leave the last enterable field (assume it is BOTMFLD), they return to the choice field, and the form is once again processed as a menu.

An argument form can also be redefined to simulate a menu. Also, any nonstandard form (that is, one without a .TYPE entry) can be processed in ALL-IN-1 by specifying an explicit form .TYPE in the FORM function call.

3.12.2 The /RESET Form Qualifier

The /RESET form qualifier is often specified in a FORM function call that contains an explicit form .TYPE parameter. For example, assume form MENARGENT contains the following Named Data entries:

Named Data

1 Name .TYPE

ARG/ PRE__FUNCTION = 'COMMAND CHECKSUM' /CLEAR/GET = TOTAL,\$TOTAL

The /RESET qualifier clears all qualifiers thus far processed, except the /GET qualifier. This feature makes the position of the /RESET qualifier significant, whereas the positions of the other qualifiers are nonconsequential (except where multiple occurrences of the same qualifier are found).

<FORM MENARGENT RESET/FIELDS = TOTAL,INPUT/GET = HEADER,\$HEADER

With the /RESET, MENARGENT is processed as an argument form with two form qualifiers:

/FIELDS, which identifies the two participating fields as TOTAL and INPUT.

/GET, which places the value of permanent symbol \$HEADER in field HEADER. (HEADER must be a display-only field since it is not among the non display-only fields activated by /FIELDS).

Assuming that the current or most recently processed form is a menu, then this function call causes the form to still be processed as a menu, with fields TOTAL and INPUT participating but without the /PRE__FUNCTION qualifier entered in Named Data. The new choice field (CHOICE2) must be specified by the /FIELDS qualifier.

3.12.3 Dynamically Redefining Entry Forms

Because an entry form's `.TYPE` and `.FILE` directives are independent of one another, only one reference form is needed to fully describe a data file. This form is the only one with a `.FILE` directive, but any number of other forms containing a `/DATA` form qualifier can access the file. For example:

```
<FORM MENENTRY ENTRY/RESET/FIELDS = TOTAL,INPUT/MODE = ADD/DATA = ACCOUNTS
```

`MENENTRY` was created as a menu but is now called as an entry form. The required `/MODE` and `/DATA` qualifiers are appended after `/RESET` clears any currently active form qualifiers.

A `WRITE` function call has a required mode parameter that overrides the current mode of the entry form (as specified in either the previous `WRITE` call or in the `.TYPE` directive in Named Data). The mode associated with an entry form determines how a data file is accessed. The `WRITE` function deals only with forms which contain the `.FILE` directive. For example:

```
<WRITE ADD ACCOUNTS %KEY = TEST, . . .
```

adds a record to the file since the form `ACCOUNTS` has a `.FILE` directive. However:

```
<WRITE ADD MENENTRY %KEY = TEST, . . .
```

does not work because the form does not have a `.FILE` directive. Instead, it has a `/DATA` qualifier which points to the form containing the `.FILE` directive.

3.13 Multipage Form Sets

A form set is a segmented or multipage form.

Define a form set by entering the `.FORM__SET` directive in the Named Data of the first page of the set, specifying the names of the remaining forms, or pages, in the set as arguments. `.FORM__SET`s cannot be defined/redefined dynamically. `.FORM__SET` syntax is:

```
.FORM__SET = form[,form ... ]
```

The first page containing the `.FORM__SET` directive must also contain the `.TYPE` directive and all Named Data entries that apply to subsequent pages. The form `.TYPE` you specify on the first page applies to the entire set. `ALL-IN-1` does not process the Named Data of the secondary pages in a form set; you can specify either forms with no Named Data entries or forms that can stand on their own if called separately.

Menus may not be built as form sets; all other form types may.

NOTE: *You can duplicate field names from page to page, but `ALL-IN-1` copies the contents of the field on the second page into the field on the first page. To avoid this, use the `|COPY` field qualifier on the first field to fill in the second field. See Section 4.9.*

When calling an entry form set in `INQUIRE` mode, the user must press Gold TAB to see subsequent pages of the form set, instead of pressing `RETURN` at the prompt.

For example, the Named Data of `FORMA` could contain the following:

Named Data

```
1 Name .TYPE
ENTRY/MODE=UPDATE
```

```
2 Name .FORM__SET
FORMB,FORMC
```

The Named Data of both `FORMB` and `FORMC` could contain the following:

Named Data

```
1 Name .TYPE
ENTRY
```

When `FORMA` is called, it displays. When the user presses `TAB` or the Down Arrow in the last field of this first page, the second page (or segment), `FORMB`, displays. `FORMB` may be a full-sized (23-line) screen or a partial overlay. How much of the screen is occluded by `FORMB` depends on the form-wide attributes, assigned through `FMS` when the form is created.

Input is solicited from the first enterable field on FORMB. This continues until the last field on FORMC, the third and last page in the set. As with FORMB, how FORMC displays depends on form-wide FMS attributes. Pressing TAB or the Down Arrow in the last field of FORMC simply causes an end-of-form message. Conversely, a BACKSPACE or Up Arrow in the first field of the second or subsequent page causes the previous page to display.

Gold TAB and Gold BACKSPACE are NEXT PAGE and PREVIOUS PAGE functions, respectively. When pressed in any field on a form-set page:

- Gold TAB displays the next page of the set and places the cursor in the first enterable field on that page.
- Gold BACKSPACE displays the previous page of the set and places the cursor in the last enterable field on that page. list

Use form sets when:

- The fields of a form do not fit on the screen at one time.
- Less critical fields need to be put on a second page that can be easily accessed but usually does not need to be displayed.
- You need to define large data records on a .FILE form with .FORM__SET and use smaller single-page forms to update subsets of a record.

3.14 Nonstandard Forms

Any form that does not contain a .TYPE entry is considered a nonstandard form. If a nonstandard form in an accessible form library is called as is with a FORM function call, an error message displays. However, if the FORM function call specifies a form .TYPE, the nonstandard form is typically processed as that form type. The SHOW__FORM function also displays nonstandard forms.

Field Processing

4.1 Introduction

Fields are volatile areas on forms that serve as information display areas or as transaction input areas. Forms serve as the primary interface between a user and ALL-IN-1, and the fields in a form are where the actual interchange of data takes place.

The Field Processing facility is responsible for managing the flow of data into and out of enterable fields. This facility processes both the information that passes through fields and the interpretation of special keys that the user presses while in a field.

The Form Processing facility usually directs Field Processing. When you call the **FORM** function, Form Processing calls Field Processing automatically to handle the interchange of information.

4.2 The FIELD Function

The **FIELD** function invokes the Form Processing facility to activate one or more enterable fields and to process the associated qualifiers in Named Data. Calling the **FIELD** function is equivalent to calling the **FORM** function with the **/FIELDS** qualifier. The syntax of the **FIELD** function is:

FIELD [field name [,field-name . . .]]

where the optional field-name parameters are the FMS names of one or more enterable (FMS non display-only) fields on the current or most recently processed form.

When you specify the **FIELD** function without parameters, **ALL-IN-1** activates all participating fields on the form. The effect is the same as if you had called the function with the wildcard parameter, *****.

4.3 Defining Fields

The creator of a form defines the size, length, position, order of access, and some of the validation characteristics of fields through the Forms Management System. **ALL-IN-1** fully supports these FMS-defined field attributes — for example, display-only, response required, must fill, upper case, justification, and numeric/alphanumeric.

ALL-IN-1 supplements these FMS attributes with field qualifiers that are handled within **ALL-IN-1** rather than in FMS. To associate one or more field qualifiers with a field, the name element of Named Data must contain the name of the field to which the qualifiers apply; the data element contains the qualifiers for the field. If the data entry is longer than 80 characters, you can extend it by repeating the field name in another Named Data name element and continuing the qualifier(s) in the associated data field.

Use field qualifiers to:

- Get information to display in a field.
- Store information from a field in a file or symbol.
- Clear a field.
- Structure a field for mathematical expressions.
- Specify pre and postprocessing of a field.

The syntax for field qualifiers is identical to the syntax for form qualifiers:

`/qualifier[ = arg][/qualifier[ = arg]. . .]`

where

`the /` is required syntax.

`qualifier` is any valid field qualifier.

`= arg` is a field qualifier argument, which may be optional.

You can abbreviate all field qualifiers to their shortest unique representation, but because DIGITAL reserves the right to add new keywords, abbreviations may become ambiguous. For this reason it is best to use the full, unabbreviated names of the qualifiers.

The following examples of valid field qualifiers are from the Named Data of form DOCDSC:

Named Data

1 Name FOLDER

`/GET__SAVE = FOLDER/PUT__SAVE = FOLDER/RECOG = CAB$FOLDER`

2 Name FILNAM

`/PUT__SAVE = $FILNAM1`

NOTE: *When a field attribute associated with an ALL-IN-1 qualifier conflicts with an FMS attribute specified by the form's creator, the FMS attribute usually overrides the ALL-IN-1 qualifier. For example, if you use the /OPTIONAL qualifier on a field that has been given the FMS Input required attribute, the user must still fill the field with valid data before continuing. Similarly, if you designate a field display-only when you create it, there is no way to solicit direct user input from the field.*

The exception is the OA\$FLD_EXIT (EXIT SCREEN) function: EXIT SCREEN overrides some FMS attributes such as input required and must fill. EXIT SCREEN and other special function keys are discussed later in this chapter.

4.4 Types of Field Qualifiers

Field qualifier types are:

- **Preprocessing Qualifiers** — Preprocessing qualifiers are processed before the form is displayed, before the user enters any fields. Preprocessing qualifiers are used to set up the form just prior to the user gaining control. Like all field qualifiers, they apply only to the participating fields specified by the `/FIELDS` form qualifier.

The preprocessing field qualifiers are distinct from the preprocessing form qualifiers. Both types of qualifiers are processed before the user enters any fields, but the field qualifiers are specific to one field in a form, while form qualifiers may be form-wide.

- **In-Field Qualifiers** — `ALL-IN-1` acts upon in-field qualifiers during field processing, either as the user enters or leaves a field or when they press a function key.
- **Validation Qualifiers** — In addition to the preliminary field validation performed by FMS (such as checking for numeric or non-numeric data), you can specify qualifiers that provide another complete level of validation or recognition. Use these validation qualifiers to check the user's entry against:
 - A key in an indexed data file
 - Any other field in a data set
 - A record selection expression (RSE)
 - A special validation expression

When the user does not enter a field qualified by a `/VALID`, `/RSE_VALID`, `/INVALID`, `/RSE_INVALID`, or `/FILE_CHECK`, `ALL-IN-1` still checks the contents of the field during postprocessing. The effect is the same as if the user entered the field.

- **Postprocessing Qualifiers** — Postprocessing qualifiers are processed after the user leaves the last active field on the form. As with all other field qualifiers, they apply only to those fields specified by the `/FIELDS` form qualifier.

Descriptions of the field qualifiers follow.

4.5 /AUTO

Use this qualifier in conjunction with the /VALID and /RECOGNITION qualifiers to enable autorecognition, or automatic field completion. When a field is qualified with /AUTO, pressing a standard terminator key (RETURN, TAB, and so on) is equivalent to pressing the CHOICE key, Gold L.

/AUTO works as follows: the user enters something in the field and then presses a terminator. If the user's entry does not match the validation expression, the Key Entry facility checks for a partial match. If the user's entry matches the first part of exactly one record, then that record field is copied to the field. If more than one match is found, a form displays listing the matching records and the user is prompted for the desired entry.

/AUTO is a validation qualifier.

4.5.1 Examples

Assume the Named Data of form PHSEL contains the following entry:

Named Data

2 Name SELDIR

/GET__SAVE=\$DIRECTORY/AUTO/VALID=OA\$TABLE:"PERSONAL,ALL-IN-1,CORPORATE"

NOTE: Do not use /AUTO in conjunction with /INVALID. Automatic field completion with /INVALID value is confusing.

4.6 /BLANK

Use /BLANK in conjunction with the /GET__SAVE qualifier. After /GET__SAVE gives a value to the field, the /BLANK qualifier suppresses all subsequent /GET__SAVEs. Thus, the first value assigned is a permanent field value, analogous to an FMS-assigned default value (and overriding any actual FMS default value for that field).

Only if a /BLANK qualified field is blank, is the /GET__SAVE processed. /BLANK is only useful on CALC and ENTRY forms. On CALC forms the user can loop through the form more than once and the fields must be

cleared after each loop is complete. On all other forms but ENTRY forms the field must be blank when the /GET__SAVE is processed (unless there is an FMS default).

/BLANK is a preprocessing qualifier.

4.6.1 Example

The Named Data of form TODOENT contains the following entry:

Named Data

```
1 NAME AI_PRIORITY  
/GET__SAVE = "A1"/BLANK/HARD = "Item Priority"
```

```
2 NAME AI_STATUS  
/HARD = 'Status of item'/GET__SAVE = "0"/BLANK
```

4.7 /CLEAR

/CLEAR clears the qualified field before processing begins. Use /CLEAR, for example, to clear a result field in CALC forms, which repeats the field-processing cycle many times.

/CLEAR is not needed on entry forms since entry form processing automatically clears all fields before each cycle. Also, /CLEAR contradicts /GET__SAVE, which loads a value in the field; and these two qualifiers should not be used on the same field.

/CLEAR is a preprocessing qualifier.

4.7.1 Examples

Named Data

```
1 Name FULLFIELD  
/CLEAR
```

This has the same effect as the /CLEAR form qualifier with field FULLFIELD as an argument.

4.8 /COMPUTE

This qualifier causes an algebraic expression to be evaluated and the result is stored in the field. The syntax for /COMPUTE is:

/COMPUTE = "expression"

where expression is any arithmetic expression. Enclose the expression in single or double quotes.

/COMPUTE is a postprocessing qualifier.

4.8.1 Examples

Assume the Named Data of calc form INVEST contains the following entries:

Named Data

1 Name PMT

/COMPUTE = "P*(1+(I/N/100))^(N*Y)"

2 Name PMT

/DECIMAL = 2

The values in fields P, I, N, and Y are combined as illustrated, and the result is stored in field PMT.

NOTE: To set the number of decimal places for a /COMPUTE result use the /DECIMAL field qualifier and the DECIMAL function. /DECIMAL, covered later in this chapter, sets the number of places for the current field only. The DECIMAL function sets the number of decimal places for all subsequent mathematical computations.

4.9 /COPY

/COPY copies the contents of a field into another field, a symbol-table symbol or both. The syntax is:

/COPY = [valfield],[field],[symbol] [; . . .]

where

valfield is the optional name of a field in a validation file that is copied to the specified destination as the second and third arguments. The valfield argument is meaningless unless a validation qualifier such as /VALID or /RECOGNITION is also specified on the same field.

At least one of the additional arguments (field, symbol) must be specified when valfield is present.

field is a field on the current form. When valfield is provided, the value of valfield is stored in field. When field is the first /COPY argument, the contents of the qualified field are copied to the specified field.

symbol is a symbol-table symbol which is assigned the value of the first argument in the list, valfield or field. If symbol is a permanent symbol, a DCL global symbol is also created (or updated) and given the same value and name (less the \$) as symbol.

All arguments are positional: at least one must be specified, and placekeeping commas must be inserted if an argument is omitted. More than one argument list may be provided by separating each group with a semicolon.

Since entry forms do implicit copies from file fields to form fields at the beginning of the data entry cycle, /COPY is generally not needed on entry form fields.

/COPY is an in-field qualifier.

4.9.1 Examples

Named Data

2 Name F1

/COPY = ,F2

3 Name F3

/VALID = PRINTER/COPY = SM__PRT__DESC,HOLDS__DESC,\$HOLDS__DESC/OPTIONAL

4 Name F4

/VALID = FORMAT/COPY = DESC,, \$HOLDS__FORMAT__DESC;AUTHOR,HOLDS__AUTHOR

5 Name F5

/COPY = ,F6,\$HOLDS__F5__AND__F6

The /COPY on Named Data line 2 copies the contents of field F1 to field F2. The comma preceding F2 identifies it as the second (field) argument and not a validation field.

Field F3 (line 3) is validated against the PRINTER master file. If a valid printer destination is entered in F3, then the contents of RMS field SM__PRT__DESC from OA\$DATA:PRNTTYPE.DAT are copied to FMS field HOLDS__DESC and also to permanent symbol \$HOLDS__DESC. Global DCL symbol HOLDS__DESC is created (or updated).

Field F4 in line 4 is validated against the FORMAT master file. If a valid format is entered in F4, then the contents of RMS field DESC from OA\$DATA:FORMAT.DAT are copied to permanent symbol \$HOLDS__FORMAT__DESC. Global DCL symbol HOLDS__FORMAT__DESC is created (or updated). Also, field AUTHOR from the FORMAT file is copied to FMS field HOLDS__AUTHOR.

On line 5, the /COPY stores the value of the qualified field (F5) both in field F6 and in permanent symbol \$HOLDS__F5__AND__F6. Global DCL symbol HOLDS__F5__AND__F6 is created (or updated).

NOTE: *Fields are copied right after the user moves off the field and during form postprocessing, while symbols are copied during form postprocessing. Thus, if a function called through a /POST_FUNCTION field qualifier references any symbol assigned with /COPY on any field, that symbol is undefined or does not contain the most recently assigned value. For example:*

Named Data

4 Name F4

/VALID=FORMAT/COPY=DESC,,\$HOLDS__FORMAT__DESC;AUTHOR,HOLDS__AUTHOR

5 Name F5

/COPY=,F6,\$HOLDS__F5__AND__F6

6 Name F5

/POST_FUNCTION="GET #TEMP=\$HOLDS__FORMAT__DESC"

The /POST_FUNCTION-specified GET on \$HOLDS__FORMAT__DESC does not load #TEMP with the value of FORMAT.DESC (assigned on field F4) because the /COPY does not take effect until the entire form is postprocessed.

4.10 /DECIMAL

Use this qualifier in conjunction with the /COMPUTE qualifier to define the number of decimal places to display when the computed value is stored in the field. The syntax is:

/DECIMAL=n

where n may be:

- The number of decimal places to display
- F, for floating-point notation
- I, for rounded integers

/DECIMAL is a postprocessing qualifier.

4.10.1 Examples

Assume the Named Data of calc form INVEST contains the following entries:

Named Data

```
1  Name  PMT
/COMPUTE = "P*(1+(I/N/100))^(N*Y)"
```

```
2  Name  PMT
/DECIMAL = 2
```

The values in fields P, I, N, and Y are combined as illustrated, and the result is stored in field PMT. The /DECIMAL field qualifier specifies the number of decimal places in the result. Also see the /COMPUTE field qualifier earlier in this chapter.

NOTE: /DECIMAL sets the number of decimal places for a single /COMPUTE expression. After the expression is evaluated, any decimal setting previously specified by the DECIMAL function is restored.

4.11 /DOC

This qualifier creates a unique VMS file name from the contents of the symbol or symbols specified as argument(s). The file name is stored in the qualified field. The syntax is:

```
/DOC [ =[basis__symbol] [,default__values__symbol] ]
```

where

basis__symbol is an ALL-IN-1 symbol containing file-root naming information. The first nine non-special characters are concatenated and the result is used as the file root.

When basis symbol is omitted or if the stripped file-root combined with the contents of default__values__symbol duplicates an existing file name, then a unique file name is created based on the current date and time. Default file names consist of 9-character alphabetic sequences derived from the unique date/time numeric sequence.

default__values__symbol is an ALL-IN-1 symbol containing default values to use in creating the file name. The default__values__symbol argument may be any full or partial RMS file specification. A dummy file-root may be inserted for clarity but it is ignored since the root is created by /DOC.

If default__values__symbol is omitted, the file name is created without a file-type extension.

Special symbol OA\$DEFAULT__EXTENSION is commonly used as the default-values argument. This symbol is a file-type extension derived from the EDITOR__STYLE field in the User Profile. The recognized editor styles and their default extensions are listed in Table 6-5; the supported editor styles and their associated extensions are listed below.

<i>Editor__Style</i>	<i>OA\$DEFAULT__EXTENSION</i>
EDT style, default editor	.TXT
WPS style, default editor	.TXT
WPS-PLUS	.WPL
other	.TXT

Like the MAKE__FILE__NAME function, /DOC simply creates unique file names; it does not create files. The qualifier calls MAKE__FILE__NAME routines and builds file names according to the same rules as that function. The routine saves the last unique name created and does not duplicate it, even when two or more /DOC qualifiers appear on the same form.

NOTE: /DOC recognizes and translates logical names in the default-values argument only when they have been defined in the main process. Logicals defined only in the subprocess are not recognized.

4.11.1 Examples

The following /DOC combinations are valid:

```
1 Name FILNAM
/DOC = USERNAME,"KUDZU::SYSS$DISK:[DAVID.OA].TXT"
```

If field **USERNAME** contains **WORCESTERSHIRE**, then **/DOC** creates the file-name root **WORCESTER**. Because the **default-values** argument is present, **WORCESTER** is expanded to:

```
KUDZU::SYS$DISK:[DAVID.OA]WORCESTER.TXT
```

Directory **KUDZU::SYS\$DISK:[DAVID.OA]** is then searched for an existing file named **WORCESTER.TXT**. If none is found, the expanded file description is stored in field **FILNAM**. Otherwise, a unique file name is built using the context information supplied, and this value is stored in **FILNAM**.

Assuming that this form is processed in the usual manner and that file **WORCESTER.TXT** is actually created, then when this form is called again, if field **USERNAME** still contains **WORCESTERSHIRE**, **/DOC** creates a file name like the following:

```
KUDZU::SYS$DISK:[DAVID.OA]ZPPAIRDHU.TXT
```

where **ZPPAIRDHU** is **/DOC**'s response when it encounters an existing file name, **WORCESTER.TXT**.

```
/DOC = $NEWUSER, #FULL__FILE__SPEC
```

If **#FULL__FILE__SPEC** has the value **"SYS\$DISK:[MARIAH]XXXX.TXT"** and symbol **\$NEWUSER** contains **"O'FLAHERTY"**, then the full file-spec is expanded to:

```
SYS$DISK:[MARIAH]OFLAHERTY.TXT
```

Directory **SYS\$DISK:[MARIAH]** is then searched for an existing file named **OFLAHERTY.TXT**. If none is found, the expanded file description is stored in the qualified field. Otherwise, a date/time-based default file name is created, and this value is stored in the qualified field. As mentioned earlier, the file-name root in the file-spec, **XXXX**, is ignored and may be omitted if desired.

```
/DOC = USERNAME, ".FLB"
```

If field **USERNAME** contains **CHINLE**, then the default directory (usually **OAUSER**) is searched for an existing file named **CHINLE.FLB**. If none is found, the expanded file description is stored in the qualified field. Otherwise, a unique, date/time-based file name is built, and this value is stored in the qualified field.

```
/DOC = "NODE::DEV:[DIR]ROOT.EXT", OA$DEFAULT__EXTENSION
```

Despite its punctuation, the first argument is treated as a literal string. The colons, brackets, and period are stripped out, leaving NODEDEVDI as the file root. This root is concatenated with the contents of OA\$DEFAULT__EXTENSION to create the expanded file-spec.

The following file-spec and all similar ones are invalid:

```
/DOC = USERNAME,KUDZU::XXX.TXT
```

Unless XXX is a logical name assigned to a device:[directory] combination, RMS discards this file specification as invalid when the /DOC-created USERNAME is expanded.

/DOC is a postprocessing qualifier. The "/INVALID=OA\$DIR" validation expression is another way to ensure that the contents of a field can be used to create a unique file name.

4.12 /FILE__CHECK

This validation qualifier assumes the field contains a VMS file name or file-name root and checks whether the file actually exists in storage. The syntax is

```
/FILE__CHECK = default-spec-symbol [,exp-name-symbol-dest]
```

where

default-spec-symbol is an ALL-IN-1 symbol that represents a default file extension, directory, device, and node that replaces the contents of the field before FILE__CHECK validates it. The user may enter any or all parts of the file specifications in the qualifier field and the default fills in the missing specifications, if any. The default-spec-symbol stores the expanded file name if /FILE__CHECK finds the file. The full syntax of default-spec-symbol is:

```
node::device:[directory]filename.extension
```

Symbol-table symbols are recognized, but quoted strings may not be because some file specifications, such as those that include a password on a remote node, can include quoted strings. You do not have to enclose literals in quotes for this qualifier.

When this argument is omitted, the default directory (usually OAUSER) is searched for a file name that corresponds to the contents of the qualified field.

exp-name-symbol-dest is the (optional) name of a symbol which receives the expanded file name.

`/FILE__CHECK` works as follows:

- 1 `ALL-IN-1` takes the contents of the qualified field and looks for special file-spec delimiters—double colons, colons, square brackets, periods, or semicolons—and treats them as such. If no delimiters are found, the field contents are treated as a file-name root. Wildcard characters and recognized and logical names are translated.
- 2 After the expanded file specification is built, storage is searched to ensure that the file exists. The search stops if the file exists or if the file cannot be found.
- 3 If the contents of the field cannot be expanded into a valid file name or if `/FILE__CHECK` cannot find the file, a message displays on line 24.

NOTE: *`/FILE__CHECK` recognizes and translates logical names in the qualified field and in both arguments only when they have been defined in the main process.*

4.12.1 Examples

If a user enters login on field `FILENAM` on form `ANYFORM`, and the Named Data of `ANYFORM` contains the following entry:

Named Data

```
1 Name FILENAM
/FILE__CHECK = KUDZU::DISK$:[JOE].COM
```

login expands to `KUDZU::DISK$:[JOE]login.COM`. `ALL-IN-1` then searches storage for the file. If the user enters `.COM` in `FILENAM` and the Named Data of `ANYFORM` contains:

```
1 Name FILENAM
/FILE__CHECK = KUDZU::DISK$:[JOE]LOGIN
```

the results are the same (although this is not a common application of the qualifier). A practical application is to search the OAUSER and OA\$LIB directories for a file; for example:

```
1  Name  FILENAM  
/FILE__CHECK=OAUSER:*.COM,OA$LIB:*.COM
```

If the contents of FILENAM do not contain any special characters, it is treated as a file-name root for a .COM file, and the two specified directories are searched for that file.

Like /VALID, /FILE__CHECK requires input. The qualified field may not be left blank unless it is also qualified by /OPTIONAL.

Validation qualifiers /RECOGNITION and /VALID offer similar capabilities with the OA\$DIR special expression, while /INVALID=OA\$DIR ensures that the contents of a field can be used to form a unique file name. These qualifiers are covered elsewhere in this chapter.

4.13 /GET__SAVE

/GET__SAVE retrieves the contents of any ALL-IN-1 symbol and places it in the field. The syntax is:

```
/GET__SAVE = symbol
```

Various symbol combinations are possible, as illustrated in the examples that follow.

/GET__SAVE is similar to the /GET form qualifier. Both qualifiers get values for the field, but /GET__SAVE overrides /GET because field preprocessing is done after form preprocessing, and /GET__SAVE is a preprocessing qualifier. /GET is used to load values into display-only fields.

NOTE: *Since menus normally only have a single enterable field (the choice field), you normally use the /GET form qualifier instead of /GET__SAVE to preload values into fields.*

4.13.1 Examples

From entry form DOADBENT:

Named Data

```
2 Name FOLDER
/GET__SAVE=$WPD0C:30

3 Name DOCNUM
/GET__SAVE=$WPD0C:6:30

4 Name TITLE
/GET__SAVE=CAB$.TITLE[$WPD0C]
```

The /GET__SAVE on Named Data line 2 retrieves the first 30 characters stored in permanent symbol \$WPD0C and loads them into field FOLDER. The following Named Data entry loads field DOCNUM with the next 6 characters of \$WPD0C (for example, the 6 characters starting from character 30, zero-offset). The final /GET__SAVE on line 4 loads the TITLE field of the specified record from the file-cabinet index file into field TITLE on this form, DOADBENT. This final example is a data set reference.

From argument form DOCSL:

Named Data

```
2 Name SELECT
/GET__SAVE=#SELECT
```

This /GET__SAVE retrieves the value of temporary symbol #SELECT and loads it into field SELECT.

Named Data

```
6 Name MODDAT
/GET__SAVE=OA$DATE:2:6 OA$DATE:2 OA$DATE:2:3/BLANK

7 Name CRETIME
/GET__SAVE=OA$TIME:2 OA$TIME:2:3/BLANK

10 Name CREDAT
/GET__SAVE="000000"/BLANK
```

Named Data line 6 illustrates how parts of three symbols can be combined with /GET__SAVE. Any type of symbol can be concatenated with any other type; the three pieces here all happen to be substrings of one special symbol, OA\$TIME. Field CRETIME is also loaded with combined parts of OA\$TIME, while field CREDAT is loaded with a quoted string.

Another example of /GET__SAVE on a quoted string/DSR combination:

```
6 Name MODDAT
/GET__SAVE = "Last Revised: " DOCDB.MODDAT[OA$CURDOC]
```

4.14 /HARD

Use the /HARD qualifier to display information about a field while in Hard Copy Mode. The syntax is:

```
/HARD = "description"
```

where "description" is a quoted string that is displayed when the user enters the field.

Since FMS forms do not display on hard-copy terminals, the concepts of field and form are not as meaningful as on a VT-series screen display. When ALL-IN-1 runs in Hard Copy Mode, the user must try to navigate without screen displays that illustrate lengths, field context, and other helpful information. The /HARD qualifier provides a line of information about a field that can help guide the user through a form. See Appendix B for more discussion of Hard Copy Mode.

4.14.1 Examples

Named Data

```
1 Name FOLDER
/VALID = DOCDB/SHOW = DOCNAM/HARD = "FOLDER: This is the key field, 60
```

```
2 Name FOLDER
characters long. Recognition is active"
```


When ALL-IN-1 runs in Hard Copy, or Command Mode, this description displays. Then the user is prompted for input into field FOLDER. The /HARD form qualifier performs the same function for forms.

/HARD is a preprocessing qualifier.

4.15 /INVALID

/INVALID compares the contents of the qualified field against the value of a validation expression. If the contents of the field match the expression, the user must enter another value in the field. The syntax is:

`/INVALID = val_exp [:post-fun[|post-fun. . .]] [/USE__FORM]`

where

`val_exp` is a validation expression, one of several types covered in the following list.

`post-fun. . .` are one or more optional ALL-IN-1 functions separated from the validation expression with a semicolon and from one another with backslashes. You cannot specify post functions if you specify a function or function list as the validation expression.

`/USE__FORM` is an optional subqualifier that you can add to control the output of /INVALID.

The validation expression may be:

- One or more expanded data-set references. The syntax is a variation of the standard DSR syntax.

`/INVALID = data-set[:key] [.data-set[:key] . . .] [.field]`

where

- | | |
|----------|---|
| data-set | is the name of a data set, usually an entry form associated directly with a data file through a <code>.FILE</code> directive. If you specify an entry form name and it contains a <code>/DATA</code> qualifier pointing to another entry form, the <code>/INVALID</code> does not work. |
| key | is any field in the data set to be used as an index into the set. If the data set is an entry-form/data-file combination, key must be an RMS key of reference. When you create the data file with an FDL description, you explicitly identify the RMS key(s) of reference. In all other cases, the RMS key(s) of reference corresponds to the key field that you specify on the <code>.FILE</code> line in the entry form's Named Data. |
| field | is another field in the data set against which ALL-IN-1 compares the contents of the <code>/INVALID</code> -qualified field. If the data set is an entry-form/data-file combination, field can be another FMS field on the form. You can specify this optional argument either with an alternate key or with the implied primary key. |

You can specify more than one data-set/key combination, but you can only specify one field. Thus, when you include a field name, all specified data sets must contain that field.

When you specify just a data set, `/INVALID` searches the set using the primary key and validates against that same field.

- A validation data set reference. You can specify several special validation DSRs with validation qualifiers. Specialized DSRs are covered, where applicable, elsewhere in this APR, and Appendix C lists all data sets.
- Any ALL-IN-1 function or list of functions, or a single script command. Precede the first function with an angle bracket (`<`) to identify the validation expression as a function (list).

For example:

```
/INVALID = <.IF OA$DIR:"*.*.%WHOLE[field] EQS " "  
THEN OA$VAL__SET__VALID ELSE Display That filename  
already exists.
```

***NOTE:** When you specify a function or function list as the validation expression, you cannot specify post-functions. Also, when you specify a function list, you must include the special function, OA\$VAL__SET__VALID to determine that the data is correct.*

Also see the /RSE__INVALID qualifier, which is similar to invalid but accepts an RSE as the validation expression.

When /INVALID is present, users cannot exit the field until they enter a unique value. That is, the contents of the qualified field must *not* match the criteria of the validation expression. This qualifier is the inverse of /VALID.

/INVALID is useful on entry forms used to add new records to a data file since it ensures that a unique key is created.

/INVALID implies the /RECOGNITION qualifier, so Recognition is active on the qualified field. However, you can also specify /RECOGNITION in conjunction with /INVALID. That is, pressing the CHOICE keys displays a list that matches the /RECOGNITION criteria, while ALL-IN-1 validates the actual contents of the field against a different validation expression.

When you use /INVALID alone, the list of invalid choices that displays when users press the CHOICE keys is either the primary keys from the validation file or a collection of matching records created by another validation expression. ALL-IN-1 outputs this list by default through form AUTO. You can override this default by specifying the /USE__FORM qualifier.

The /OPTIONAL field qualifier overrides the input required feature of /INVALID. However, anything entered in the field must still meet the validation criteria.

4.15.1 Examples

Named Data

```
1 Name  AI__PRIORITY
/INVALID = OA$TABLE:"0, +, - "
```

```
2 Name  CURUDP
/PUT__SAVE = $CURUDP/INVALID = OA$DIR:"[.UDP]*.UDP"
```

These two Named Data entries illustrate /INVALID with special DSRs. If users enter 0, +, or - in field AI__PRIORITY, a warning message displays and they must try again. Line 2 illustrates how you can use /INVALID to ensure that ALL-IN-1 creates a unique file name. The OA\$DIR DSR flags as invalid any field entry that duplicates an existing file name when expanded to [.UDP]file-name.UDP. The /DOC and /FILE__CHECK field qualifiers offer related capabilities.

4.16 /LIST

This qualifier identifies a selection-list file. The syntax is:

```
/LIST = filename__symbol
```

where the single, required argument is a quoted string or symbol-table symbol that equates to a VMS file name. By convention, selection lists have the file-type extension .SEL. Selection lists are used by the NEXT__LIST and the OA\$FLD__NEXT__LIST functions. As illustrated in Table 4-2, an entry in the Named Data of form DEFAULT associates the keypad period key with the OA\$FLD__NEXT__LIST function. Whenever you press this key or call OA\$FLD__NEXT__LIST in some other way while on the current field, ALL-IN-1:

- 1 Opens the file identified by /LIST
- 2 Selects the next entry (or the first entry, if the file is being used for the first time)
- 3 Loads that entry into the qualified field

Selection lists are created by select forms with a special select-form qualifier, also named /LIST. These lists are sequential ASCII files that you can edit like any other text file. Five asterisks identify the original top of

the list, and the next record is always the current first line in the file. For example, assume FORMATS.SEL contains the following entries:

```
BLISS
COBOL
*****
FORTRAN
PASCAL
VALGOL
```

BLISS is the next record and is selected the next time /LIST references this FORMATS.SEL and NEXT__LIST or OA\$FLD__NEXT__LIST opens this file. BLISS is then placed at the end of the file, and COBOL becomes the next item. Selection lists remain stable from session to session, so the next selection remains FORTRAN until you use FORMATS.SEL again.

/LIST is an in-field qualifier.

4.16.1 Examples

```
4 Name SELNAME
/OPTIONAL/LIST = #SELFIL/RECOGNITION = #DFORM/AUTO/SHOW = FULNAM
```

```
5 Name FORMAT
/RECOG = $FORMAT/LIST = "TOPLIST.SEL"
```

If the user enters field SELNAME and presses the NEXT LIST key, ALL-IN-1 calls OA\$FLD__NEXT__LIST to load the next entry in file #SELFIL into SELNAME. If the user enters field FORMAT and presses NEXT LIST, ALL-IN-1 calls OA\$FLD__NEXT__LIST to inject the next entry in TOPLIST.SEL into FORMAT.

4.17 /OPTIONAL

Use this qualifier in conjunction with /VALID, /INVALID, /RSE__VALID or /RSE__INVALID to indicate that the field may be left blank. By default, /VALID and /INVALID require that the user provide input in the qualified field. /OPTIONAL overrides this default. However, ALL-IN-1 still checks the field contents against the validation expression.

/OPTIONAL is an in-field qualifier.

4.17.1 Examples

Named Data

1 Name TITLE
/INVALID = NEWFILE:NAME.TITLE/OPTIONAL

2 Name FOLDER
/VALID = \$FVALID/OPTIONAL/GET__SAVE = \$WPFDR/PUT__SAVE = \$IFOLDER

In this example, the user can leave fields TITLE and FOLDER blank.

4.18 /POST__FUNCTION

The /POST__FUNCTION field qualifier identifies one or more functions to process just after the user moves off the field in the usual way. The syntax is:

/POST__FUNCTION = 'function [function. . .]'

where backslashes are required to separate one function call from the next when you specify a function list of more than one function.

When you specify a function list, enclose the list in single or double quotes. Complex function lists often include combinations of quoted strings, so the choice of quote characters both within and around the function list becomes important. Quotes are optional with single /POST__FUNCTION functions.

/POST__FUNCTION is identical in syntax and similar in behavior to the /POST__FUNCTION form qualifier. For each field that is qualified with /POST__FUNCTION, if the user dismisses the form in the usual way, all specified functions are always executed. It does not matter if the user has entered a given field. This feature allows field postprocessing on fields that may fill in automatically (for example, with the /COPY qualifier). The differences between the field and form /POST__FUNCTION are:

- When the user enters a field, the /POST__FUNCTION field qualifier only dispatches its functions when the user leaves that field normally. When you press EXIT SCREEN, ALL-IN-1 ignores /POST__FUNCTION-specified functions. In contrast, the /POST__FUNCTION form qualifier dispatches its functions regardless of how you leave the form.

- ALL-IN-1 processes the /POST__FUNCTION field qualifier just as the user moves off the field (except in the case covered previously, when the user does not enter a given field); it processes the form qualifier just as the user moves on to the next form.

4.18.1 Examples

Named Data

```
1 Name UNAME  
/ POST__FUNCTION='COMMAND NAMECHECK'
```

In this example, the COMMAND function invokes NAMECHECK.COM if the user leaves this field normally (for example, without an EXIT SCREEN). ALL-IN-1 does not execute /POST__FUNCTION field-qualifier functions if the user presses EXIT SCREEN to leave the field.

4.19 /PRE__FUNCTION

The /PRE__FUNCTION qualifier specifies one or more functions to perform every time the user moves onto the field and before anything is entered in the field. The syntax is:

```
/PRE__FUNCTION='function [\function ...]'
```

where backslashes are required to separate one function call from the next when more than function is specified.

When you specify a function list, enclose the entire list in single or double quotes. Complex function lists often include combinations of quoted strings, so the choice of quote characters both within and around the function list becomes important. Quotes are optional with single /POST__FUNCTION functions.

/PRE__FUNCTION is identical in syntax and similar in behavior to the /PRE__FUNCTION form qualifier. The only difference is that ALL-IN-1 processes the form qualifier as soon it calls the form, while it processes the field qualifier just as the user enters the field.

4.19.1 Examples

Named Data

7 Name SELNAME

```
/PRE__FUNCTION='GET #SELFIE="PH" $PH__SHORT__DIR "LIST.SEL"
```

8 Name SELNAME

```
\GET #DFORM="PH" $PH__SHORT__DIR "ENT"__\GET $PHNUMBER=" "
```

9 Name SELNAME

```
\GET $PH__PER__NUM=" " "
```

In this example, /PRE__FUNCTION dispatches four separate GET functions as the user enters field SELNAME. The first GET builds a combination of two quoted strings and a permanent symbol into local symbol #SELFIE. The next GET builds local symbol DFORM in the same way. The last two GET function calls clear two permanent symbols, \$PHNUMBER and \$PH__PER__NUM.

NOTE: Unlike the /POST__FUNCTION field qualifier, /PRE__FUNCTION is only executed when the user touches the field. Thus, qualifying a display-only field with /PRE__FUNCTION is meaningless, and any fields that have not been activated are not processed.

4.20 /PROMPT

This qualifier displays a line of text when the user enters the qualified field. ALL-IN-1 clears the prompt when the user leaves that field. The syntax is:

```
/PROMPT=[field,]"text"
```

where

field is the optional name of a field where the text displays; if omitted, the prompt displays on line 24 of the screen.

"text" is the text of the prompt, a quoted string.

4.20.1 Example

Named Data

```
1 Name  AFIELD
/VALID = FORMAT/PROMPT = 'Enter a format such as "RUNOFF": '
```

When the user enters AFIELD, a prompt displays on line 24 explaining what they should enter in the field. Both types of quote characters are used here to avoid confusion.

4.21 /PUT__SAVE

/PUT__SAVE stores the contents of the qualified field in one or more symbols. The syntax is:

```
/PUT__SAVE = symbol-name [,symbol-name. . .]
```

where ALL-IN-1 treats symbol-name as a symbol-table symbol. ALL-IN-1 also creates global DCL symbols in the subprocess for each permanent symbol in the /PUT__SAVE argument list.

If a specified symbol does not exist, ALL-IN-1 creates it.

NOTE: As a postprocessing qualifier, /PUT__SAVE is done only after the form is processed, and then only if the user exits the form normally (for example, without an EXIT SCREEN). The /POST__FUNCTION qualifier is processed before /PUT__SAVE. Thus, you can modify the contents of a field through a /POST__FUNCTION call and /PUT__SAVE a different value than the user enters. You cannot save symbols with /PUT__SAVE and then reference them in a /POST__FUNCTION call.

4.21.1 Examples

Named Data

```
7 Name  SELNAME
/GET__SAVE = $AAA/PUT__SAVE = $AAA
```

```
8 Name  NEWNAME
/PUT__SAVE = #BBB,$CCC,$DDD
```

`/GET__SAVE` preloads field `SELNAME` with the value of permanent symbol `$AAA`. If the user exits the form in the usual way, `/PUT__SAVE` puts the new contents of `SELNAME` back into `$AAA`. In addition, `/PUT__SAVE` creates global DCL symbol `AAA` for the life of the subprocess.

The `/PUT__SAVE` on field `NEWNAME` saves the contents of that field in three symbols: local symbol `#BBB`, and permanent symbols `$CCC` and `$DDD`. `/PUT__SAVE` also creates global DCL symbols `CCC` and `DDD` for the life of the subprocess.

4.22 /RECOGNITION

This qualifier activates the ALL-IN-1 Recognition features in the qualified field. The user can press the CHOICE keys, Gold L and Gold S to view a list of matching entries. They can select one entry, by number, from this list, which `/RECOGNITION` displays on form `AUTO`. If you also specify the `/AUTO` field qualifier, form `AUTO` automatically displays all partially matching entries when the contents of the field are ambiguous.

The syntax is:

```
/RECOGNITION = val__exp [:post-fun[\post-fun. . .]] [/USE__FORM]
```

where

`val__exp` is a validation expression, one of several types covered in the following list.

`post-fun. . .` are one or more optional functions to execute if the contents match the validation expression. Separate the functions from the validation expression with a semicolon and from one another with backslashes. Do not specify post functions if you specify a function or function list in the validation expression.

`/USE__FORM` is an optional subqualifier that you can add to control the output of `/INVALID`.

The validation expression may be:

- One or more expanded data-set references. The syntax is a variation of the standard DSR syntax:

```
/RECOGNITION = data-set[:key] [,data-set[:key]...] [.field]
```

where

data-set is the name of a data set, usually an entry form associated directly with a data file through a **.FILE** directive. If you specify an entry form name and the form contains a **/DATA** qualifier pointing to another entry form, **/RECOGNITION** does not work.

key is any field in the data set to be used as an index into the set. If the data set is an entry-form/data-file combination, key must be an RMS key of reference. When you create the data file with an FDL description, you explicitly identify the RMS key(s) of reference. In all other cases, the RMS key(s) of reference correspond to the key field or fields that you specify on the **.FILE** line in the entry form's Named Data.

field is another field in the data set against which **ALL-IN-1** compares the contents of the **/RECOGNITION** qualified field. If the data set is an entry form/data file combination, field can be another FMS field on the form. You can specify this optional argument either with an alternate key or with the implied primary key.

You can specify more than one data-set/key combination, but only one field. Thus, when you include a field name, all specified data sets must contain that field.

When you specify just a data set, **/RECOGNITION** searches the data set using the primary key and validates against that same field.

- A validation data set reference. You can specify several special validation DSRs with validation qualifiers. Specialized DSRs are covered, where applicable, elsewhere in this APR, and Appendix C lists all data sets.

- Any ALL-IN-1 function or list of functions, or a single script command. Precede the first function with an angle bracket (<) to identify the validation expression as a function (list). The syntax is:

`/RECOGNITION = <function[\function. . .]`

NOTE: When you specify a function or function list as the validation expression, you cannot specify post-functions.

Although `/VALID` and `/INVALID` imply `/RECOGNITION`, either of these qualifiers can coexist with `/RECOGNITION`. That is, pressing the CHOICE keys displays a list that matches the `/RECOGNITION` criteria, while ALL-IN-1 validates the actual contents of the field against a different validation expression.

4.22.1 Examples

```
NAMEDDATA INDEX=2 NAME='VMSFILE'  
  DATA='/PUTSAVE = VMSFILE/GETSAVE = VMSFILE/RECOG = OA$DIR:";" ' ;  
  
END_OF__FORM NAME='EDTGETVMS' ;
```

Pressing Gold L or Gold S in field `COMPILE` invokes the `OA$DIR` DSR to display a list of command procedures in the current default directory.

From the Named Data of form `CXP$HOSTS`:

```
16 HOSTNAME  
/RECOG = CXP$HOSTS.HOSTNAME/AUTO/SHOW = HOST__DESC
```

Pressing Gold L or Gold S in field `HOSTNAME` displays a list of the values for that field (`HOSTNAME`) in the `CXP` host file.

4.23 /RSE__nnn

Three validation qualifiers allow you to use a record selection expression (RSE) while validating and/or activating Recognition in a field. An RSE is a combination of data sets, fields, symbols, and/or quoted strings. The qualifiers that recognize RSEs as arguments are `/RSE__VALID`, `/RSE__INVALID`, and `/RSE__RECOGNITION`.

All three of these qualifiers have the same syntax:

`/RSE__nnn = rse [;post-fun[\post-fun. . .]] [/USE__FORM]`

where

`/RSE__nnn` is either:

- `/RSE__VALID`
- `/RSE__INVALID`
- `/RSE__RECOGNITION`

`rse` is a record selection expression.

The syntax is identical to that for FOR-function RSEs. During field validation, ALL-IN-1 transforms the validation expression into a FOR function call before checking the field contents.

`post-fun. . .` are one or more optional functions to execute if the contents match the RSE. Separate the functions from the RSE with a semicolon and from one another with backslashes.

`/USE__FORM` is an optional subqualifier that you can add to control the output of the qualifier.

These three qualifiers are covered in the following paragraphs.

`/RSE__INVALID`

This qualifier serves the same purpose as `/INVALID`. It compares the contents of the qualified field against the value of a validation expression. If the contents of the field match the expression, the user must enter another value in the field.

`/RSE__RECOGNITION`

This qualifier serves the same purpose as `/RECOGNITION`: it activates the ALL-IN-1 Recognition features in the qualified field. The user can press the CHOICE keys, Gold L and Gold S to view a list of matching entries. They can select one entry, by number, from this list, which `/RSE__RECOGNITION` displays on form AUTO. If you also specify the `/AUTO` field qualifier, form AUTO automatically displays all partially matching entries when the contents of the field are ambiguous.

/RSE__VALID

This qualifier serves the same purpose as **/VALID**. It compares the contents of the qualified field against the results of a validation expression. If the contents of the field match the expression, **/RSE__VALID** accepts the input.

The major difference between direct FOR-function calls and RSEs in **/RSE__INVALID**, **/RSE__VALID**, and **/RSE__RECOG** is that while the FOR function always specifies a DO action function, **/RSE__nnn** qualifiers never do. Instead, as covered earlier, you may optionally append a list of one or more functions to the RSE. Separate these functions from one another with backslashes and from the RSE with a semicolon.

The **UNIQUE** keyword causes FOR to select the first matching record if there are several that meet the criteria. You can achieve the same effect while validating a field with the **/UNIQUE** field qualifier.

ALL-IN-1 only executes post-functions if the contents of the field match the validation expression—or in the case of **/RSE__INVALID**, does *not* match the validation expression. Also, when you append post-functions, **ALL-IN-1** does not load selected key values into the qualified field. You must do this from within the function list.

4.23.1 Examples

From the Named Data of form AIP:

```
8 AI__CLASS
/HARD = 'Item Class'/GET = $AI__CLASS/RSE__RECOG = AIE.AI__CLASS WITH .AI__TYPE
```

```
9 AI__CLASS
EQS "A" AND .AI__CLASS ENS AI__CLASS
```

```
10 AI__CLASS
/SHOW = " " " .AI__ITEM " " " .AI__DESC'
```

Pressing Gold L or Gold S in field **AI__CLASS** invokes Recognition with an RSE. **/RSE__RECOG** displays all records in data set AIE with an A in the FMS **AI__TYPE** field and RMS field **AI__CLASS** equal to FMS field **AI__CLASS** on form AIP. **/SHOW** formats the display.

4.24 /SCROLL

/SCROLL identifies the qualified field as a scrolled region, a line or series of lines that scroll to allow the entry and display of more information than can fit in the area at one time. /SCROLL either associates the scrolled region with a data file or calls one or more functions as the user moves around in the region. This qualifier also activates several scrolling features within the field. The syntax is:

`/SCROLL = [max-symb],[file-symb], [start], [data-set]`

where

max-symb is the maximum number of lines (records) in the scroll file. The limit is 2000 and is the default if this argument is omitted.

file-symb is an ALL-IN-1 symbol representing an indexed file that /SCROLL creates or updates through the scrolled region. If you omit this argument, the qualifier looks for an indexed file with the default name of SCROLL.DAT. If /SCROLL does not find the file, it creates it.

This argument is only meaningful when you do *not* specify a data set argument.

start is the starting record number that /SCROLL loads in the scrolled region. When omitted, start defaults to 1.

data-set is any valid data set with two fields, in the following format:

NUMBER — key field, 4 characters long; contains an ASCII numeral that corresponds to the line number of an item.

TEXT — 132 characters of data that displays in the qualified scrolled region.

The default data set is SCROLL; it corresponds to file OAUSER:SCROLL.DAT, which /SCROLL creates if it does not exist. You can specify other data sets, including SCROLL\$FUNCTION and several Calendar scrolling data sets.

Field Processing

All these arguments are positional, and you must insert placekeeping commas if you omit one or more arguments.

NOTE: To qualify a field with */SCROLL*, you must have created it as a scrolled region with *FMS*. See the *VAX FMS* documentation set for information on creating scrolled regions within forms.

/SCROLL automatically redefines the following keyboard keys and associates them with special scroll functions to control scrolling:

Key Sequence	Function	Description
UP	OA\$SCL_UP	Scroll up 1 line
DOWN	OA\$SCL_DOWN	Scroll down 1 line
GOLD TAB	OA\$SCL_NEXT_PAGE	Next page
GOLD BACKSPACE	OA\$SCL_PRIOR_PAGE	Prior page
GOLD T	OA\$SCL_FIRST_PAGE	First page
GOLD B	OA\$SCL_LAST_PAGE	Last page
GOLD UP	OA\$SCL_TOP	Top of scrolled region
GOLD DOWN	OA\$SCL_BOTTOM	Bottom of scrolled region
RETURN	OA\$SCL_RETURN	Down 1 line or next field

/SCROLL automatically defines these keys when the cursor enters a scrolled region and automatically resets them to their original values for the form when the user exits the region.

For example, form XYZ contains the following Named Data entries:

Named Data

```
1 Name AAA
/POST_FUNCTION = "Display The next field is a scrolled region"
```

```
1 Name BBB
/SCROLL = , $SCROLL__FILE
```

Tabbing from field AAA to field (or region) BBB results in seven internal keyboard assignments that associate the preceding key sequences with OA\$SCL__ function calls. Thus, pressing the Down Arrow in BBB calls OA\$SCL_DOWN.

4.24.1 /SCROLL with Indexed Files

When you do not specify a data set argument, /SCROLL treats the second parameter as an ALL-IN-1 symbol representing the name of an indexed file containing the data to be displayed in the scrolled region. If the file does not exist, /SCROLL creates it with the following main attributes:

FILE

BUCKET_SIZE	2
NAME	"value of file-symb argument"
ORGANIZATION	indexed

RECORD

FORMAT	fixed
SIZE	137

KEY 0

DUPLICATES	no
INDEX_AREA	0
NULL_KEY	no
PROLOGUE	3
SEG0_LENGTH	5
SEG0_POSITION	132
TYPE	string

/SCROLL treats the 5-character field at position 132 as the key field. This field always contains a line number corresponding to the position of the record in the scroll file. The line number in this key field corresponds to the /SCROLL start-rec argument (if any). /SCROLL also uses the value internally as an argument for the OA\$SCL_ functions that move the cursor around in the scrolled region.

The other field in the scroll-file record is simply the text, 132 characters long.

If you specify a filename without an extension, /SCROLL automatically appends a .DAT.

When you first call a form containing `/SCROLL=[max], 'filename', [start]`, `/SCROLL` first checks whether the specified file already exists. If it does, `/SCROLL` opens the file and loads as much of it as possible into the scrolled region, beginning with line 1 — or with line start, if this parameter is specified. If the file does not exist, `/SCROLL` creates it and names it filename.

When the user enters the scrolled region, the key definitions listed previously take effect. Any changes to an existing file overwrite the previous contents of the file, and any entries into a new file are written to the file exactly as entered. The `/SCROLL` activity ceases when the user presses TAB, BACKSPACE, RETURN, or EXIT SCREEN to leave the field.

4.24.2 SCROLL\$FUNCTION

Scrolling is page-based, where a page corresponds to the current contents of the scrolled region. Each page is `p` lines long, where `p` is the number of lines in the scrolled region. Thus, `OA$SCL__NEXT__PAGE`, for example, replaces the current contents of the scrolled region with `p` new lines of text, beginning with the line after the last line of the current page. Press Gold TAB to call `OA$SCL__NEXT__PAGE`.

You always know the value of `p` because you can always see the number of lines in the scrolled region. The `SCROLL$FUNCTION` special DSR makes other, dynamically changing information available to a specified function. This information includes the number of the current line, the contents of that line, and the key sequence that you pressed to move off that line.

When you qualify a field with `/SCROLL=,,,SCROLL$FUNCTION:"function"`, function is called when the user first enters the scrolled region and during scrolling whenever the user moves off the current field — for example, whenever an `OA$SCL__` function is called. You can specify any ALL-IN-1

function or function list as the `SCROLL$FUNCTION` argument, and `SCROLL$FUNCTION` communicates with the functions it calls in these ways:

- `SCROLL$FUNCTION` appends two parameters—code and line—to the function(s) as follows: specifying

```
/SCROLL = ,,SCROLL$FUNCTION:"function"
```

results in `SCROLL$FUNCTION` dispatching the function call:

```
function code,line
```

where code is a number corresponding to the action being performed by `/SCROLL` and line is the line number on which that action is being performed. `/SCROLL` stores the contents of the text for the line in permanent symbol `$SCROLL__TEXT`.

- Permanent symbol `$SCROLL__TEXT` acts as a shuttle, ferrying lines of text back and forth between the called function and the scrolled region. Whenever the user enters something in a scrolled region, the entry is stored in `$SCROLL__TEXT`. If `SCROLL$FUNCTION`-dispatched function alters the contents of this permanent symbol, `/SCROLL` reloads the new value into the line that contained the original value.

Each call to an `OA$SCL__` function triggers at least one, and sometimes two, calls to the `SCROLL$FUNCTION` function(s). Each `OA$SCL__`-triggered call is associated with preset code values. Each code value in turn corresponds to an action taken during scrolling. These `SCROLL$FUNCTION` actions are summarized in the following table.

Table 4-1 `SCROLL$FUNCTION` Actions

Scrolling Function	Action Description	Resulting Calls to Function
OA\$SCL__UP	After leaving line n:	
	If entering line n-1 means scrolling up off the top of the page, OA\$SCL__UP calls:	function 9,n
	and remains on line n.	function 4,n-1

(continued)

Table 4-1 SCROLL\$FUNCTION Actions (Cont.)

Scrolling Function	Action Description	Resulting Calls to Function
OA\$SCL__DOWN	After leaving line n:	function 9,n
	If entering line n+1 means scrolling down off the bottom of the page, OA\$SCL__DOWN calls:	function 4,n+1
	and remains on line n.	
OA\$SCL__NEXT PAGE	After leaving line n:	function 9,n
	For each line y, where y ranges from t+p to t+2p, where t is the line number of the first line in the current page, and p is the number of lines in the scrolled region, OA\$SCL__NEXT_PAGE calls:	function 4,y
	before stopping at line t+p.	
OA\$SCL__PRIOR_PAGE	After leaving line n:	function 9,n
	For each line y, where y ranges from t-2p to t-p, where t is the line number of the first line in the current page and p is the number of lines in the scrolled region, OA\$SCL__PRIOR_PAGE calls:	function 4,y
	before stopping at line t-2p.	
OA\$SCL__FIRST_PAGE	After leaving line n:	function 9,n
	For line y, where y ranges from 1 to p, where p is the number of lines in the scrolled region, OA\$SCL__FIRST_PAGE calls:	function 4,y
	before coming to rest in line 1.	

(continued)

Table 4-1 SCROLL\$FUNCTION Actions (Cont.)

Scrolling Function	Action Description	Resulting Calls to Function
OA\$SCL_LAST_PAGE	After leaving line n:	function 9,n
	For line y, where y ranges from max-p to max, where max is the number of records in the scroll file and p is the number of lines in the scrolled region, OA\$SCL_LAST_PAGE calls:	function 4,y
OA\$SCL_BOTTOM	before stopping at line max.	
	After leaving line n:	function 9,n
	For line y, where y ranges from max-p, where max is the number of lines in the scrolled region and p is the current line in the scrolled region, OA\$SCL_BOTTOM calls:	function 4,y
	before stopping at line max.	

The function codes are explained in the following text.

SCROLL\$FUNCTION appends one of the following code numbers with the current line number. Each code number corresponds to an action taken by /SCROLL.

1 – OPEN

Code 1 indicates that the scroll file has been opened by OA\$SCL_INIT. It remains open until closed by OA\$SCL_EXIT.

2 – DEACCESS

Code 2 indicates that the scroll file has been closed by OA\$SCL_EXIT.

4 — GET__BY__KEY

Code 4 indicates that line y has been retrieved from the scroll file and will be loaded into the current line in the scrolled region. This code is returned by several OA\$SCL__ functions when they move into a scrolled line that is being displayed for the first time. The corresponding line in the scroll file is retrieved and loaded into the scrolled region.

9 — PUT__RECORD

Code 9 indicates that line y has been retrieved from the scrolled region. This code is generated by OA\$SCL__ functions as they move off the current line. It serves as a flag to the SCROLL\$FUNCTION function to write the line back into the scroll file, in case user input altered the line.

10 — UPDATE

Code 10 is identical to code 9.

As mentioned earlier, permanent symbol \$SCROLL__TEXT always contains either the current contents of the scrolled line or some text that the SCROLL\$FUNCTION called function has loaded into that symbol for subsequent action by scroll.

4.24.3 Example

The following example shows the processing performed for a form with /SCROLL=,,,SCROLL\$FUNCTION:"COMMAND SCRTEST" on a scrolled region.

- 1 When the form is first called, the file is opened and each line in the scrolled region is filled. If the beginning field of the form is not the scrolled field then the file is closed.

```

! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 1,0
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,2
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,3
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,4
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,5
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,6
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,7
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,8
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,9
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,10
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,11
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 2,0

```

- 2 When you enter the scrolled field the file is opened.

```

! ZOA-I-LOGFUN, Function: OA$FLD_DOWN
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 1,0

```

- 3 Moving in the scrolled region first writes the current line and then requests any necessary data for new lines.

```

! ZOA-I-LOGFUN, Function: OA$FLD_DOWN
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,1
! ZOA-I-LOGFUN, Function: OA$FLD_DOWN
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,2
! ZOA-I-LOGFUN, Function: OA$FLD_UP
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,3
! ZOA-I-LOGFUN, Function: OA$FLD_PAGE_FOR
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,2
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,12
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,13
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,14
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,15
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,16
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,17
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,18
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,19
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,20
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,21
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,22

```

```

! ZOA-I-LOGFUN, Function: OA$FLD_PAGE_BAC
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,12
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,2
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,3
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,4
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,5
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,6
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,7
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,8
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,9
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,10
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,11
! ZOA-I-LOGFUN, Function: OA$FLD_TOP
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,1
! ZOA-I-LOGFUN, Function: OA$FLD_BOTTOM
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,1
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1990
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1991
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1992
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1993
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1994
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1995
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1996
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1997
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1998
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1999
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,2000
! ZOA-I-LOGFUN, Function: OA$FLD_TOP
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,1990
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,1
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,2
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,3
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,4
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,5
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,6
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,7
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,8
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,9
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,10
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 4,11

```

- 4 When you leave the scrolled region, the current line is written to the file and then the file is closed.

```

! ZOA-I-LOGFUN, Function: OA$FLD_NEXT
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 9,1
! ZOA-I-LOGFUN, Function: COMMAND          SCRTEST 2,0
! ZOA-I-LOGFUN, Function: OA$FLD_DONE

```


4.24.4 Customized Scrolling

Display-only fields cannot be scrolled by the default mechanisms which ALL-IN-1 provides. You must use the `OA$SCL_SET_FIELD` function before enabling scrolling in a display-only field. This function sets up the context of the field so that the scrolling keys may be used. The syntax is:

`OA$SCL_SET_FIELD [max-symb], [file-symb], [start], [data-set],
field-name`

where

max-symb is the maximum number of lines (records) in the scroll file. The limit is 2000 and is the default if this argument is omitted.

file-symb is an ALL-IN-1 symbol representing an indexed file that /SCROLL creates or updates through the scrolled region. If you omit this argument, the qualifier looks for an indexed file with the default name of SCROLL.DAT. If /SCROLL does not find the file, it creates it.

This argument is only meaningful when you do *not* specify a data set argument.

start is the starting record number that /SCROLL loads in the scrolled region. When omitted, start defaults to 1.

data-set is any valid data set with two fields, in the following format:

NUMBER — key field, 4 characters long; contains an ASCII numeral that corresponds to the line number of an item.

TEXT — 132 characters of data that displays in the qualified scrolled region.

The default data set is SCROLL; it corresponds to file `OAUSER:SCROLL.DAT`, which /SCROLL creates if it does not exist. You can specify other data sets, including `SCROLL$FUNCTION` and several Calendar scrolling data sets.

field-name is the name of the field on which the scrolling takes place.

You must insert placekeeping commas if you omit one or more optional arguments.

An example which illustrates display-only scrolling in form WEEKSH follows:

```
1 Name DESC
/SCROLL = 1440,,#CAL__START,CAL$SCROLL__WEEK__SCH

2 Name .UP
OA$SCL__SET__FIELD 1440,,1,CAL$SCROLL__WEEK__SCH,DESCIOA$SCL__TOIOA$SCL__UP

3 Name .UP
IOA$FLD__STAY

4 Name .GOLD DOWN
OA$SCL__SET__FIELD 1440,,1,CAL$SCROLL__WEEK__SCH,DESCIOA$SCL__BOTTOM

5 Name .GOLD DOWN
IOA$FLD__STAY
```

The following sequence shows what happens when this form processes.

- 1 During initialization of the form, the /SCROLL qualifier causes the scrolled region to fill. Then, when a user presses the Up Arrow key (or any of the scrolling keys defined on the form) a series of functions are called.
- 2 The first function called is OA\$SCL__SET__FIELD, which establishes the context for the remaining functions.
- 3 OA\$SCL__TOP combines with OA\$SCL__UP and causes the scrolled field to scroll up one line when a user presses the Up Arrow key (or any of the scrolling keys defined on the form).
- 4 OA\$FLD__STAY keeps the cursor on the form. The scrolling keys are terminator keys and cause the form to exit without the OA\$FLD__STAY function.

4.25 /SHOW

Use this qualifier in conjunction with /INVALID, /RSE__INVALID, /VALID, /RSE__VALID, /RECOGNITION, or /RSE__RECOG to identify fields from a record collection and control how they are output. You can also use /SHOW on the key fields of entry forms.

Use /SHOW to:

- Format a record collection built by a validation expression or extracted directly from an entry form. The syntax is:

```
/SHOW = valfield[:length] [,valfield[:length]. . .]
```

where

valfield is a field on the entry form associated with the validation file.

Length is the number of characters of that field to display. There is always one implicit space between fields, and when the specified length exceeds the number of characters in the field, /SHOW pads the display with blanks.

Although you can validate against more than one file at a time without using an RSE, you can only specify one field, and all files must share that field. Thus, /SHOW's valfield argument must be a shared field if more than one file is involved.

Recognition is automatically enabled in key fields of entry forms, and this syntax is used when you specify /SHOW on these fields.

- To further control the format of record collection, enclose the formatting string in single or double quotes. The syntax is:

```
/SHOW = ".valfield[:length] .valfield[:length]. . .]"
```

Precede each valfield by a dot to identify it as a field in the validation file(s) (as opposed to a field on the current form). Also, separate the fields with spaces instead of commas.

This syntax is identical to that of the SEL__DISPLAY subfunction of the FOR function. There are no implicit spaces, and you must specify spaiing between fields. For example:

```
/SHOW = '.valfield[:length] " " .valfield[:length]. . .]'
```

This syntax actually expresses a symbol list, and any ALL-IN-1 symbol can be specified along with validation fields. Examples of symbol lists are given under the GET function.

/SHOW-formatted records display in the following instances:

- When the user presses the CHOICE keys in a validated field
- When the user presses the CHOICE keys in a key field of an entry form (where recognition is automatically enabled)
- When you have qualified a field with the /AUTO field qualifier (assuming that the field is also qualified with /INVALID, /RECOGNITION, or /VALID)

The primary purpose of /SHOW is to provide context information to the user to help in identifying which choice to select.

/SHOW is an in-field qualifier.

4.25.1 Examples

From form FCGETDX:

```
2 Name NUMBER  
/OPTIONAL/VALID = WPDECDX:DOCNUM.DOCNUM/SHOW = NAME,MODDAT
```

4.26 /STORE

The /STORE qualifier is unique to entry forms. Entry forms describe data files, and the fields on the form usually correspond to fields in the file, both in name and in size. /STORE retrieves the value of an alternate field from the file and stores it in the qualified field on the form. /STORE allows you to use entry forms as windows on data files that do not match the form's field layout.

The syntax is:

/STORE = alternate-field

where alternate-field is an RMS field in the entry-form-related data file.

4.27 /UNIQUE

Use /UNIQUE in conjunction with /VALID, /INVALID, and /RECOGNITION. When ALL-IN-1 searches a data set using a non-unique alternate key, /UNIQUE causes just the first occurrence of that record to be selected. /UNIQUE is to validation qualifiers what the UNIQUE keyword is to the FOR function.

/UNIQUE limits the search to one match per file. If you include more than one file /UNIQUE limits the search to one match per file for each file that is specified.

4.27.1 Example

From form WPINDEX

```
1 Name FOLDER
/UNIQUE/USE__FORM = ./PUT__SAVE = #SEARCH__FOLDER/HARD = "Folder"
```

4.28 /USE__FORM

Use this qualifier in conjunction with a validation qualifier to specify an alternate form on which to display choices.

The syntax is:

```
/USE FORM = form-name
```

4.29 /VALID

/VALID compares the contents of the qualified field against the results of a validation expression. If the contents of the field match the expression, /VALID accepts the input. The syntax is:

```
/VALID = val__exp [;post-fun[!post-fun. . .]] [/USE__FORM]
```

where

val__exp is a validation expression, one of several types covered in the following list.

`post-fun...` are one or more optional functions to execute if the contents match the validation expression. Separate the functions from the validation expression with a semicolon and from one another with backslashes. You cannot specify post functions if you specify a function or function list as the validation expression.

`/USE__FORM` is an optional subqualifier that you can add to control the output of `/VALID`.

The validation expression may be:

- One or more expanded data-set references. The syntax is a variation of the standard DSR syntax.

`/VALID = data-set[:key] [,data-set[:key]. . .] [.field]`

where

`data-set` is the name of a data set, usually an entry form associated directly with a data file through a `.FILE` directive. If you specify an entry form that contains a `/DATA` qualifier pointing to another entry form, `/VALID` does not work.

`key` is any field in the data set to be used as an index into the set. If the data set is an entry-form/data-file combination, `key` must be an RMS key of reference. When you create the data file with an FDL description, you explicitly identify the RMS key(s) of reference. In all other cases, the RMS key(s) of reference corresponds to the key field or fields that you specify on the `.FILE` line in the entry form's Named Data.

`field` is another field in the data set against which ALL-IN-1 compares the contents of the `/VALID`-qualified field. If the data set is an entry form/data file combination, `.field` can be another FMS field on the form. You can specify this optional argument either with an alternate key or with the implied primary key.

You can specify more than one data-set/key combination, but only one field. Thus, when you include a field name, all specified data sets must contain that field.

When you specify just a data set, `/VALID` searches the data set using the primary key and validates against that same field.

- A validation data set reference. You can use several special DSRs with /VALID. Specialized DSRs are covered, where applicable, elsewhere in the APR, and Appendix C lists all data sets.
- ALL-IN-1 function or list of functions, or a single script command. Precede the first function by an angle bracket (<) to identify the validation expression as a function (list). The syntax is:

/VALID = <function[\function. . .]

NOTE: *When you specify a function or function list as the validation expression, you cannot specify post-functions.*

When /VALID is present, the user cannot exit the field until a matching value is entered. That is, the contents of the qualified field must match the criteria of the selection expression. /VALID is helpful on entry forms used to update or delete existing records in a data file.

/VALID implies the /RECOGNITION qualifier, so recognition is active on the qualified field. However, you can specify /RECOGNITION in conjunction with /VALID. That is, pressing the CHOICE keys displays a list that matches the /RECOGNITION criteria, while ALL-IN-1 validates the actual contents of the field against a different validation expression.

When you use /VALID alone, the list of valid choices that displays when the user presses the CHOICE keys (or when the /AUTO field qualifier is also present) is always either the primary keys from the validation file or a collection of matching records created by some other type of validation expression. /VALID outputs the list by default through form AUTO. You can override this default by specifying the /USE__FORM qualifier.

The /OPTIONAL field qualifier overrides the must fill feature of /VALID. However, when the field contains something, /VALID still requires that the contents match the validation expression.

4.29.1 Examples

Named Data

```
1 Name CT_FLD
/VAL = <.IF CT_FLD GE "0" AND CT_FLD LE "256" THEN .FX OA$VAL__SET__VAL
```

```
2 Name TITLE
/OPTIONAL/VALID = DOCDB:TITLE/SHOW = DOCNUM/PUT__SAVE = $ITITLE
```

```
3 Name AI_STATUS
/VALID = OA$TABLE:"O,C"
```

```
4 Name SELNAME
/VALID = #DFORM.USER/COPY = PHONE,,$PHNUMBER;HOMEPH,,$PH__PER__NUM
```

```
5 Name FORMAT
/VALID = FORMAT/SHOW = DESC/PUT__SAVE = $IFORMAT/OPTIONAL
```

Named Data line 1 illustrates how you call a script command to validate the contents of a field. In this example, the contents of the field must be within the specified range. The call to the function, `OA$VAL__SET__VALID`, is required. `OA$VAL__SET__VALID` is intended for this sort of validation expression.

`/VALID` validates the contents of field `TITLE` (line 2) against the personal File-Cabinet index, `OAUSER:DOCDB.DAT`, as indexed by RMS field `TITLE`.

`/VALID` validates the contents of field `AI_STATUS` against a 2-entry table: O and C. The user must enter one of these characters in the field.

Named Data line 4 validates the contents of field `SELNAME` against the contents of field `USER` in the entry-form/optional-key combination represented by local symbol `#DFORM`. `#DFORM` must contain a value that allows this validation expression to be expanded into a valid combination.

The final example, in Named Data line 5, illustrates the most typical use of this qualifier. `/VALID` validates the contents of field `FORMAT` against the key field of the file associated with entry form `FORMAT`.

4.30 Data Validation

The general syntax for a validation qualifier is:

```
/val-qual = val__exp [:post-fun[ \post-fun . . .]] [/USE__FORM]
```

where

/val-qual may be /INVALID, /RSE__INVALID, /RECOGNITION, /RSE__RECOG, /VALID, or /RSE__VALID.

val__exp is a validation expression, one of several types covered in the following list.

post-fun . . . are one or more optional functions to execute if the contents match (or, in the case of /INVALID, do NOT match) the validation expression. Separate the functions from the validation expression with a semicolon and from one another with backslashes.

NOTE: You cannot specify post-functions when you specify a function list as the validation expression. When you specify one or more post-functions, *ALL-IN-1* does not automatically place a selected key value in the qualified field. You must ensure that at least one of the post-functions loads the qualified field with any selected value. Special symbol *OA\$SEL KEY* contains the key-value of any record that the user selects through form *AUTO*. The simplest way to put that value into a field is with a *GET* function call.

/USE__FORM is an optional subqualifier that you can add to control the output of the validation qualifier.

The validation expression may be:

- One or more expanded data set references. The syntax is a variation of the standard DSR syntax:

```
/val-qual = data-set[:key] [.data-set[:key] . . .] [.field]
```

where

data-set is the name of a data set, usually an entry form associated directly with a data file through a *.FILE* directive. If you specify an entry form that contains a /DATA qualifier pointing to another entry form, the validation qualifier does not work.

- key** is any field in the data set to be used as an index into the set. If the data set is an entry form/data file combination, key can be any FMS field on the form. The key is optional. When it is omitted, ALL-IN-1 accesses the file by the primary key with which it was created.
- field** is another field in the data set against which ALL-IN-1 compares the contents of the qualified field. If the data set is an entry form/data file combination, field can be another FMS field on the form. You specify this optional argument either with an alternate key or with the implied primary key.

You can specify more than one data-set/key combination, but only one field. Thus, when you include a field name, all specified data sets must contain that field.

When you specify just the data set, ALL-IN-1 searches the data set using the primary key and validates against that same field.

Multiple forms are OR-linked. ALL-IN-1 collects and displays all possible matches. You can modify this feature with the /UNIQUE qualifier, which limits the collection to one unique record per match, per file.

The following combinations, and others like them, are valid:

/INVALID = entry__form

/RECOGNITION = entry__form,entry__form,entry__form

/VALID = entry__form.field

/VALID = entry__form,entry__form.field

/VALID = entry__form:key.field

/VALID = entry__form:key,entry__form:key.field

/INVALID = entry__form:key.field

while a combination like the following is invalid:

/INVALID = entry__form:key.fielda,entry__form:key.fieldb

For example:

Named Data

1 Name ZZZ

/VALID = FORMAT1.DESC/OPTIONAL/SHOW = FORMAT

2 Name AAA

/VALID = FORMAT1/OPTIONAL/SHOW = DESC

3 Name BBB

/VALID = FORMAT1,FORMAT2 .DESC/OPTIONAL/SHOW = FORMAT

4 Name CCC

/VALID = FORMAT1:DESC/SHOW = FORMAT

/VALID validates field ZZZ against RMS field DESC in the file associated with form FORMAT1. DESC automatically displays when the user presses Gold S or Gold L, so the /SHOW qualifier just specifies field FORMAT.

The /VALID on field AAA is the simplest kind of validation: against the primary key in a single-file data set.

/VALID validates field BBB against RMS field DESC in the files associated with forms FORMAT1 and FORMAT2. As mentioned before, this is an OR validation; that is, if the user presses Gold L or Gold S in BBB, then ALL-IN-1 searches field DESC in both files and lists all possible matches. Specifying the /UNIQUE qualifier here limits the search to one unique occurrence of DESC for each file.

/VALID validates field CCC against RMS field DESC, and the data file associated with form FORMAT1 is also indexed by DESC. ALL-IN-1 displays the contents of this field and of field FORMAT when the user presses Gold L or Gold S in the field.

- A validation data set reference. You can specify several special validation DSRs with validation qualifiers.
- A record selection expression. RSEs are only valid with /RSE__INVALID, /RSE__RECOG, and /RSE__VALID. This is the only type of validation expression you can specify with these qualifiers.
- Any ALL-IN-1 function or list of functions, or a single script command.

NOTE: When you specify a function or function list as the validation expression, you cannot specify post-functions.

For example:

Named Data

1 Name IP__FLD

/VAL = <.IF IP__FLD GE 0 AND IP__FLD LE 4095

2 Name IP__FLD

THEN .FX OA\$VAL__SET__VALID ELSE .FX DISPLAY

3 Name IP__FLD

Initial page number limited to 0 - 4095\FORCE

4 Name PM__FLD

/VAL = <.IF PM__FLD GE 0 AND PM__FLD LE 999

5 Name PM__FLD

THEN .FX OA\$VAL__SET__VAL ELSE .FX DISPLAY Left

6 Name PM__FLD

margin limited to 0 - 999 columns\FORCE

In both cases here, /VALID calls the script .IF command to validate the contents of a field. In these examples, the user must enter values that fall within a specified range before they can move off the field. The call to OA\$VAL__SET__VALID, is required either as a THEN or an ELSE action in order to mark the contents of the field as valid. OA\$VAL__SET__VALID is intended for this sort of validation expression.

4.30.1 Combining Validation Qualifiers

Although /VALID and /INVALID are mutually exclusive, as are /RSE__VALID and /RSE__INVALID, you can combine any one of these qualifiers with /RECOGNITION or /RSE__RECOG. In addition, you can qualify a field with other qualifiers, such as /UNIQUE and /SHOW, to further refine validation.

Since combining /RECOGNITION with /VALID or /INVALID can be confusing, validation expressions must be carefully combined. Since the user is selecting a number from the /RECOGNITION-created list, you must ensure that the value that ALL-IN-1 (or you) copies back into the qualified field meets the /VALID criteria.

One solution is to create the two validation expressions so they reference the same file, but this may be a waste of resources. You could use `/VALID` or `/INVALID` by itself. Combining two qualifiers becomes useful when the key to a file corresponds exactly to a non-RMS-key field in some other file, and you need to provide more context information than one file can provide.

Combining qualifiers also provides an alternative to RSEs when you want Recognition to list only those items which match the contents of fields that the user has already filled in, and you do not need the validation qualifier to check matches with those other fields.

***NOTE:** If you combine `/RECOGNITION` or `/RSE__RECOG` with `/INVALID`, `/RSE__INVALID`, `/VALID`, or `/RSE__VALID`, then `/SHOW` always refers to fields in the Recognition validation expression.*

4.30.2 Controlling Record Collection Output

Whenever a validation expression generates a collection of records, you can use two qualifiers to control the output of those records. The primary way to output field-validation information is through the `/SHOW` field qualifier. `/SHOW` takes as its arguments the names of one or more fields to display to the user. `/SHOW` also does some basic output-record formatting.

Additional record formatting capabilities are provided by `/USE__FORM`, which you can specify as a subqualifier for `/INVALID`, `/RSE__INVALID`, `/RECOGNITION`, `/RSE__RECOG`, `/VALID`, or `/RSE__VALID`, with any validation expression. `/USE__FORM` has the following syntax:

`/USE__FORM = form`

This qualifier specifies a form on which the matching records are displayed. The form may be any form TYPE, but it must have a display-only scrolled region called DISPLAY. When you omit `/USE__FORM`, ALL-IN-1 displays matching records on argument form AUTO.

For example:

Named Data

1 Name FOLDER
`/RECOG = CAB$.FOLDER/USE__FORM = DOCRECOG`

2 Name DOCNUM2
`/RSE__RECOG = XYZ WITH .AAA EN AAA`

3 Name DOCNUM2
`/SHOW = '.AAA:20 " " .BBB:7:Z .CCC:40' /USE__FORM = .`

If the user presses Gold L or Gold S in field FOLDER, the matching entries display on form DOCRECOG. In Named Data line 2, the /USE__FORM identifies the current form as the output form. This form must have a scrolled region named DISPLAY.

The /USE__FORM qualifier is the only way to provide header information with the fields that /SHOW displays. Usually /SHOW information is displayed on form AUTO, which contains no header information. Also, /USE__FORM=., which is primarily used with select forms, enhances performance since ALL-IN-1 does not have to load and display a new form.

4.31 Default Keyboard Definitions

Memory-resident form DEFAULT contains keyboard definitions for many Gold-key sequences and terminators. The remainder of this section describes what these keys do. If there is any key you do not want available to users, remove its definition from DEFAULT.

ADDITIONAL (Gold A)

Gold A is a system-wide additional options/fields/topics key. Pressing Gold A in a menu choice field causes additional menu options to overlay the currently displayed options. Pressing Gold A in an entry form that contains additional optional fields will cause those fields to display.

NOTE: *By convention, when Gold A is used to access a series of additional options from non-menu forms, the key is redefined on each form in the set, and on the last form in the set is defined as follows:*

1 Name .GOLD A

DISPLAY Additional options not available.

BACKSPACE

By default, the BACKSPACE key calls the OA\$FLD__PREV function, which moves the cursor to the previous enterable field on the form. You define the order of entry for fields when you create a form, so the previous field is predetermined. If the user is in the first field of a form, the BACKSPACE key is invalid unless the form is the second or subsequent page of a multipage form set. In this case, BACKSPACE displays the previous page.

OA\$FLD__PREV sets OA\$FORM__DISPOSE to 3, and pressing BACKSPACE sets OA\$FORM__TERMINATOR and OA\$FIELD__TERM to 2.

BOTTOM (Gold B)

By default, Gold B calls the OA\$FLD__BOTTOM function, which moves the cursor to the last enterable field in the currently displayed form. In form sets, Gold B moves the cursor to the last field in the form set.

OA\$FLD__BOTTOM sets OA\$FORM__DISPOSE to a value of 3.

CHOICE (Gold L)

By default, Gold L (a CHOICE key) calls OA\$FLD__RECOG, which activates partial Recognition in those fields that are qualified with the validation qualifiers /VALID, /INVALID, /RECOGNITION, or /RSE__xxx, and in key fields of entry forms. Pressing Gold L usually displays a list of all the possible entries that begin with the contents of the field.

For example, field XYZ is validated against the user profile:

```
1 Name XYZ
/VALID = PROFIL/SHOW = FULNAM
```

Entering HAR in this field and then pressing Gold L will display a list of all usernames that begin with HAR: HARDING, HARVEY, etc. If there is only one valid choice that begins with HAR (say HARVEY), then /VALID automatically copies HARVEY to field XYZ.

CHOICE (Gold S)

By default, Gold S (a CHOICE key) calls OA\$FLD__SEARCH, which activates full Recognition in those fields that are qualified with the validation qualifiers /VALID, /INVALID, or /RECOGNITION, and in key fields of entry forms. Pressing Gold S usually displays a list of all the possible entries that contain the contents of the field.

For example, field XYZ is validated against the user profile:

```
1 Name XYZ
/VALID = PROFIL/SHOW = FULNAM
```

Entering HAR in this field and then pressing Gold S displays a list of all usernames that contain HAR: CHARLES, HARDING, SHARON, etc. If there is only one valid choice that contains HAR (say CHARLES), then ALL-IN-1 automatically copies CHARLES to field XYZ.

COMPUTE (Gold #)

By default, Gold # calls calc form EDCLC1. This short form accepts and evaluates a single-line formula similarly to the desk calculator, form OA\$DESK.

DCL (Gold \$)

By default, Gold \$ calls the DCL function, which transfers control to DCL in the subprocess if the user has the PRVDCL privilege in their User Profile and the subprocess is not already active.

NOTE: Gold \$ is also active while editing a document with the EDT or WPS editor, or the optional WPS-PLUS editor.

DO (RETURN or Gold F)

By default, the RETURN key and Gold F both call the OA\$FLD__DONE function, which signals successful completion of a form or task or confirms the entry of data. You may also use the DO keys to indicate a true, yes, or affirmative answer to prompts.

OA\$FLD__DONE sets OA\$FORM__DISPOSE to 2, and pressing RETURN sets OA\$FORM__TERMINATOR and OA\$FIELD__TERM to 0. If the current form contains no enterable fields, form processing calls OA\$FLD__DONE automatically, so ALL-IN-1 sets these special symbols as if the user pressed RETURN.

Down (Down Arrow)

By default, the Down Arrow calls OA\$FLD__DOWN, which moves the cursor to the beginning of the first enterable field on the next line of a form. You can redefine this internally-defined key with a .DOWN directive.

On many menus that have current items—for example, current document, current name in a directory and so on—the Down Arrow is defined to select the next item. When the last item is the current item, the next Down Arrow selects the first item.

Also see the Up Arrow.

EXIT SCRIPT (Gold X, CTRL/C)

The EXIT SCRIPT keys terminate the current script and cause the Script facility to unwind from all levels of nesting and exit. The Script facility defines Gold X and CTRL/C internally; if a script is not active, Gold X is not recognized, and ALL-IN-1 displays the default message:

```
Cannot exit from CBI from here
```

When you press Gold X or CTRL/C in a script, the Script facility first looks at special symbol OA\$SCRIPT_EXIT_HANDLER, which you can load with the name of an ALL-IN-1 function call (typically DO cleanup__script__name) that Script dispatches before returning control to the point of call.

CBIs are scripts; thus, Gold X and CTRL/C can also be considered the EXIT CBI keys.

NOTE: *You can redefine Gold X with a keypad directive, but as covered in a subsequent section, you cannot redefine CTRL/C.*

EXIT SCREEN (keypad 0, Gold K or Gold Q)

By default, keypad 0, Gold K and Gold Q call OA\$FLD_EXIT, the EXIT SCREEN function. This function means that you leave the current form with no further processing. The user is saying “I do not want to proceed”. It is the opposite of DO/RETURN/Gold F. EXIT SCREEN is used to indicate a false, no, or negative answer to prompts.

OA\$FLD_EXIT sets OA\$FORM_DISPOSE to 0, and pressing keypad 0 sets OA\$FORM_TERMINATOR and OA\$FIELD_TERM to 112.

HELP (Gold H)

By default, Gold H calls the HELP function. HELP extracts and displays text from a help library based on current context information—form name, field name, last user input (if any), keyboard information (based on form type), and the last error message. HELP also provides a list of additional options.

INTERRUPT MENU (Gold I)

By default, Gold I calls form `INTMENU`, the Interrupt Menu. This menu displays the username, the system time, and the number of messages waiting in the Inbox, and provides menu options that allow access to `DATATRIEVE` the Scratch Pad and the Desk Calculator form.

MESSAGES (Gold W)

By default, Gold W calls form `OA$VIEW__MESSAGES`, which displays the current contents of the message buffer in a scrolled region. `OA$VIEW__MESSAGES` is a VIEW form, a special `.TYPE` used to provide background information.

Only one line of a message ever displays on line 24 when `ALL-IN-1` signals an informational or error message. Pressing Gold W will show the full contents of the message buffer, including any additional messages that the last error or the last user action may have triggered.

Pressing Gold H at the `OA$VIEW__MESSAGES` prompt calls `HELP` with the ID of the error as a keyword, thus providing help on the error message if help is available.

NAMED DATA (Gold N)

By default, Gold N (the `NAMED DATA` key) calls form `OA$VIEW__ND`, which displays the Named Data of the current form or form-set in a scrolled region. `OA$VIEW__ND` is a VIEW form, a special `.TYPE` used to provide background information.

NEXT LIST (keypad .)

By default, keypad calls `OA$FLD__NEXT__LIST`. When called from a field which is qualified by the `/LIST` field qualifier, `OA$FLD__NEXT__LIST` gets the top entry in the specified selection list and puts it in the field.

NEXT PAGE (Gold TAB or NEXT SCREEN)

By default, Gold TAB or NEXT SCREEN calls the `OA$FLD__PAGE__FORWARD` function. When called from any field in a form set `OA$FLD__PAGE__FORWARD` displays the next page in the set. This function places the cursor in the first enterable field on the page. `OA$FLD__PAGE__FORWARD` is equivalent to `OA$FLD__NEXT` when the user is in the last field of a page.

OA\$FLD__PAGE__FORWARD sets **OA\$FORM__DISPOSE** to a value of 3.

PREVIOUS PAGE (Gold BACKSPACE or PREV SCREEN)

By default, Gold BACKSPACE or PREV SCREEN calls the **OA\$FLD__PAGE__BACK** function. When called from any field in a form set, **OA\$FLD__PAGE__BACK** displays the preceding page in the set. This function places the cursor in the last enterable field on the page. **OA\$FLD__PAGE__BACK** is equivalent to **OA\$FLD__PREV** when the user is in the first field of a page.

OA\$FLD__PAGE__BACK sets **OA\$FORM__DISPOSE** to a value of 3.

PRINT SCREEN (Gold P)

Gold P calls the **PRINT__SCREEN** function, which makes a copy of the current screen exactly as it is displayed (lines 1 thru 23). The destination of the output depends on the following conditions:

- If the printer port can be called (as determined by the User Profile) and the port is active (that is, the **PORT__ON** function has been called), then **PRINT__SCREEN** outputs to the printer port.
- If the user is currently in the EDT or WPS text editors, then **PRINT__SCREEN** outputs to the primary editor buffer, **PBUF0**.
- If the user is in the WPS-PLUS editor and accessing the Interrupt Menu, then **PRINT__SCREEN** outputs to the editor paste buffer.
- Otherwise, **PRINT__SCREEN** stores the output in the Scratch Pad, **OAUSER:PASTE.TMP**. If **PASTE.TMP** already exists, **PRINT__SCREEN** appends the output to it, separated by a form feed. You can include the Scratch Pad into a text file in the editor by pressing Gold G or you can access the Scratch Pad Menu from any menu by pressing Gold *, which is covered in the following paragraphs.

SCRATCH PAD (Gold *)

If the user is not in a text editor, Gold * calls form **SPMENU**, the Scratch Pad menu. This menu provides options for editing, reading, deleting, and printing RMS file **OAUSER:PASTE.TMP**, which contains the contents of the Scratch Pad.

Field Processing

If the user is in the EDT, WPS or WPS-PLUS text editor, Gold * calls form SPEDMENU. This is a version of the SPMENU form, containing all of the Scratch Pad options except the Edit option.

TAB

By default, the TAB key calls the OA\$FLD__NEXT function, which moves the cursor to the next enterable field on the form. You order fields when you create a form, so the next field is predetermined.

If the user is in the last active field of a form, OA\$FLD__NEXT is invalid unless the form is part of a multipage form set and not the last page in the set. In this case, OA\$FLD__NEXT displays the next page and places the cursor in the first field on the screen.

OA\$FLD__NEXT sets OA\$FORM__DISPOSE to 3, and pressing TAB sets OA\$FORM__TERMINATOR and OA\$FIELD__TERM to 1.

TOP (Gold T)

By default, Gold T calls the OA\$FLD__TOP function, which moves the cursor to the first enterable field in the currently displayed form. In form sets, Gold T goes to the first field in the set.

OA\$FLD__TOP sets OA\$FORM__DISPOSE to a value of 3.

On many menus that have current items—for example, current document, current name in a directory and so on—Gold T is defined to select the first item. Also see the following section on the Up Arrow.

UP (Up Arrow)

By default, the Up Arrow calls OA\$FLD__UP, which moves the cursor to the beginning of the first enterable field on the previous line of a form. You can redefine this internally-defined key with the .UP directive.

On many menus that have 'current' items—for example, current document, current name in a directory and so on—the Up Arrow is defined to select the first item. See also the Down Arrow.

VERIFY ON/OFF (Gold (and Gold))

Gold (sets DCL VERIFY and turns on the SCRIPT-function TRACE Mode. Gold) disables both of these modes. These keys are useful when you are developing applications.

VIEW (Gold V)

By default, Gold V (the VIEW key) calls form OA\$VIEW__FIELDS, which displays information about the fields on the current form or form-set in a scrolled region. OA\$VIEW__FIELDS is a VIEW form, a special .TYPE used to provide background information.

TRACE (Gold %)

By default, GOLD % (the TRACE key) calls up form TRACEARG. This form permits the application developer to turn on tracing. and to specify which of the tracing options is to be logged.

4.32 Defining the Keyboard

You override all the default definitions listed in the preceding section by specifying new definitions within Named Data of any form. You create new special-function key sequences in the same manner. Most ALL-IN-1 functions are called through user-defined keys.

When any form becomes current, the keyboard definitions, if any, in that form override those predefined in the default form (usually DEFAULT) until you (or ALL-IN-1) call a new form. ALL-IN-1 recognizes all active keyboard definitions in Named Data during FORM, FIELD, PROMPT, PROMPT_ID, WAIT, YESNO__PROMPT and YESNO__PROMPT_ID function calls. For example, since Gold V calls form OA\$VIEW__FIELDS by default, pressing Gold V at a prompt displays that form to display information about the fields on the current (or most recently processed) form.

Field Processing also associates all the following key sequences with numeric values and short mnemonics. Use these numerical values and mnemonics in scripts to simulate the pressing of a key.

To define or redefine a key sequence, simply preface its mnemonic, as listed in Appendix D, with a period; enter this directive as the Name element in Named Data of the form for which you are defining the sequence. The associated Data element is the function or function list that ALL-IN-1 executes when the user presses the key sequence.

Several examples from Named Data follow:

1 Name .BS
FORM PRVFRM

In this example, the BS mnemonic is prefaced with a period and associated with a FORM function call. Pressing the BACKSPACE key in any field of this form displays form PRVFRM.

2 Name .COMMA
DTR

In this example, pressing the keypad COMMA in any field of this form places the user in interactive DATATRIEVE.

3 Name .GOLD M
COMMAND MESSAGES

Gold X directives allow you to define keystroke combinations of the Gold key (PF1) and some other key. In this example, the keystroke sequence PF1, M executes the command procedure MESSAGES.COM.

4 Name .GOLD KEY 4
GET OA\$FUNCTION = "LIST " \$THISFILE

Gold KEY n directives, where n is any numeric keypad key, allow you to define Gold-keypad sequences. In this example, the key sequence PF1, keypad 4 lists the file named by permanent symbol \$THISFILE.

5 Name .GOLD HYPHEN
GET @OA\$FIELD__NAME = #T__VALUE

In this example, the key sequence PF1, keypad HYPHEN loads the value of temporary symbol #T__VALUE into the current field.

6 Name .KEY 0
FORM MAIN

.KEY n directives allow you to define the numeric keypad keys where n is an integer from 0 to 9. In this example, keypad 0 (EXIT SCREEN) calls the FORM function to display the Main Menu.

Please note the following restrictions when defining keys:

- If you redefine a key or key sequence, and the functionality associated with the key(s) is only accessible through that key sequence, then you can no longer use that functionality on the form that contains the new definition. For example, if you redefine Gold I on form NEWENTRY, you no longer are able to access the Interrupt menu from NEWENTRY. If you redefine Gold F, however, you can still press RETURN at any menu to dismiss the form in the usual way.
- ALL-IN-1 recognizes no differences between Gold uppercase-char and Gold lowercase-char sequences, but FMS automatically translates all Name elements (and thus all directives) to uppercase.
- Although Appendix D provides mnemonics for most or all of the keys on VT1xx and VT2xx keyboards, several of these keys are processed directly by FMS or VMS and cannot be redefined. These include:
 - Left Arrow — Gold LEFT, however, is definable.
 - Right Arrow — Gold RIGHT, however, is definable.
 - DELETE (RUB CHAR) and DELETE.
 - LINE FEED (RUB WORD) and Gold LINE FEED.
 - PF2 and Gold PF2 — This is the FMS Help key.
 - PF3 and Gold PF3 — This is the FMS Insert/Overstrike key.
 - CTRL/W and CTRL/R — FMS Refresh keys.

Other mnemonics listed in Appendix D are only meaningful in the context of ALL-IN-1 scripts. LBRACE, for example, refers to the Left Brace and is not a definable key sequence.

The symbols OA\$FIELD_TERM and OA\$FORM_TERMINATOR, are discussed in subsequent sections.

4.33 Terminator Processing

A few ALL-IN-1 functions are, by nature, interactive; they require some sort of response from the user before they complete. These interactive functions are FORM, FIELD, WAIT, PROMPT, PROMPT_ID, YESNO_PROMPT, and YESNO_PROMPT_ID. Another group of ALL-IN-1 functions, internal terminator functions, have two limited, but important, tasks:

- They move the cursor off the current field and into a new field or form.
- They set dispositions by assigning a value to special symbol OA\$FORM_DISPOSE.

These terminator functions are prefixed OA\$FLD__. Each of the interactive functions reserves a few terminator keys internally and associates those keys with terminator functions.

For example, when you call a form, ALL-IN-1 places you in the first enterable field on that form, and you stay there until you press a terminator. Even when you have removed the definitions for TAB and BACKSPACE from form DEFAULT, you can still use these two keys to move around in the form (assuming there is more than one field) because the FORM function internally associates TAB with OA\$FLD__NEXT and BACKSPACE with OA\$FLD__PREV, if they are undefined.

Similarly, FORM associates RETURN with OA\$FLD__DONE and keypad 0 with OA\$FLD__EXIT; and you can still dismiss a form with RETURN or EXIT SCREEN if you have removed all keypad definitions for these functions. In contrast, if you remove the .GOLD H directive from the Named Data of DEFAULT and do not redefine it, pressing Gold H results in the message:

```
GOLD H is undefined
```

These internal defaults are listed in Table 4-2.

Table 4-2 Control-Function Calls to Internal Terminator Functions

Control Function	Calls:	When User Presses:	Side Effects:
FORM and FIELD	OA\$FLD_DONE	RETURN	1
	OA\$FLD_NEXT	TAB	1
	OA\$FLD_PREV	BACKSPACE	1
	OA\$FLD_EXIT	Keypad 0	1
	OA\$FLD_PAGE_BACK	Gold BACKSPACE	1
	OA\$FLD_PAGE_FORWARD	Gold TAB	1
	OA\$FLD_BOTTOM	Gold B	1
	OA\$FLD_TOP	Gold T	1
	OA\$FLD_NEXT_LIST	Keypad period	
	OA\$FLD_DOWN	Down Arrow	
	OA\$FLD_UP	Up Arrow	
PROMPT, PROMPT_ID, OA\$FLD_DONE		RETURN	1,2
WAIT, YESNO_PROMPT, OA\$FLD_EXIT		Keypad 0	1,2
YESNO_PROMPT_ID			

Side Effects:

- 1 —Sets OA\$FORM_DISPOSE to the value given in Table 4-3.
- 2 —Sets DCL symbols RESULT and TERMINATOR when called from the subprocess. RESULT contains any input the user entered at the prompt, and TERMINATOR contains the same value as OA\$FORM_TERMINATOR and OA\$FIELD_TERM.

These internal defaults can always be overridden with a normal keypad directive in Named Data. However, if you do override a default, ensure that some other key calls the function that the overridden terminator used to call.

As mentioned in Table 4-2, all terminator functions assign a value to `OA$FORM__DISPOSE`. This value is the disposition, or current status, of the form. The dispositions are:

- **STAY** — Functions that set the STAY disposition are those that, when called, do not cause the form to go away. `OA$FLD__NEXT`, `OA$FLD__PREV`, `OA$FLD__EXIT`, `OA$FLD__PAGE__BACK`, `OA$FLD__PAGE__FORWARD`, `OA$FLD__BOTTOM`, `OA$FLD__TOP`, `OA$FLD__UP` and `OA$FLD__DOWN` all set the STAY disposition.

These functions assign a value of 3 to `OA$FORM__DISPOSE`.

- **DONE** — Functions that set the DONE disposition are those that, when called, dismiss the form normally. `OA$FLD__DONE` sets the DONE disposition and assigns a value of 2 to `OA$FORM__DISPOSE`.
- **EXIT** — Functions that set the EXIT disposition are those that, when called, dismiss the form without further processing. `OA$FLD__EXIT` sets the EXIT disposition and assigns a value of 0 to `OA$FORM__DISPOSE`.

Since `OA$FORM__DISPOSE` is always odd for the STAY disposition and always even for DONE and EXIT, a true value always indicates that the user wishes to stay on the form and a false value always indicates that the user wishes to exit.

Table 4-3 summarizes dispositions and the `OA$FORM__DISPOSE` values associated with them.

Table 4-3 Terminator Functions, Dispositions, and `OA$FORM__DISPOSE`

Function	Disposition	<code>OA\$FORM__DISPOSE</code>
<code>OA\$FLD__EXIT</code>	EXIT	0
<code>OA\$FLD__DONE</code>	DONE	2
<code>OA\$FLD__BOTTOM</code>	STAY	3
<code>OA\$FLD__DOWN</code>	STAY	3
<code>OA\$FLD__NEXT</code>	STAY	3
<code>OA\$FLD__PAGE__BACK</code>	STAY	3
<code>OA\$FLD__PAGE__FORWARD</code>	STAY	3
<code>OA\$FLD__PREV</code>	STAY	3
<code>OA\$FLD__TOP</code>	STAY	3
<code>OA\$FLD__UP</code>	STAY	3

ALL-IN-1 saves the numeric value of the key. It is a terminator in an interactive function stored in special symbols OA\$FORM__TERMINATOR and OA\$FIELD__TERM. The terminator value is the value given in Appendix D for each key. These are always associated with the given key and not with any function the key might call.

Some of the most commonly used terminators have numeric values that differ from the corresponding OA\$FIELD__TERM value. These exceptions are listed in Table 4-4.

Table 4-4 OA\$FORM__TERMINATOR Values

Mnemonic	OA\$FORM__TERMINATOR
BS	2
TAB	1
CR	0
RETURN	0
UP	17
DOWN	16
KEY 0	112

4.34 Field Processing

When the FORM function opens a form, ALL-IN-1 builds a field definition table that describes all of the fields on the form. ALL-IN-1 marks the active fields: these are the fields that participate while the form displays. The /FIELDS form qualifier, if present, identifies the active fields. When /FIELDS is not present, all fields are activated.

The Form Processing facility controls Field Processing. For example, the argument form (.TYPE ARG) calls Field Processing but does nothing with the results above and beyond what Field Processing itself does. Entry forms (.TYPE ENTRY) first use Field Processing to get the key value(s) needed to retrieve a record from a data file, and then to update the record. The menu form (.TYPE MENU) uses Field Processing to obtain input from the choice field and any other enterable fields, and then uses the contents of the choice field to determine the ALL-IN-1 function to call.

The following summarizes Field Processing:

- ALL-IN-1 builds and scans a field table:
 - Field Processing identifies active fields that contain preprocessing field qualifiers — /CLEAR, /GET__SAVE, and /BLANK. Field Processing processes these qualifiers.
 - If a scan of the field table shows no active fields marked as enterable (that is, if there are no participating fields that the user can input into), post-processing begins. Form Processing sets special symbols OA\$FORM__TERMINATOR to 0 and OA\$FORM__DISPOSE to 2, as if the user had pressed the RETURN key.
- Form Processing uses the /BEGIN form qualifier to determine the starting field to place the cursor for entry. If there is no /BEGIN, then the first active, enterable field on the form (non display-only field) is the starting point.
- Field Processing processes the starting field. If a /PROMPT qualifier is present, the text specified in the /PROMPT displays. If the /HARD qualifier is present and the user is in Hard Copy Mode, the hard copy description displays. If the /PRE__FUNCTION qualifier is present on this field, ALL-IN-1 dispatches the function(s) before the user enters the field. Form Processing now displays the form if it is not already the current form.
- The user presses some kind of terminator key or key sequence to signal the end of input, and Field Processing processes this terminator as covered in the preceding section.

5

Generic Menu Design

This chapter discusses the design and constraints of the generic menu which ALL-IN-1 uses. Adherence to this specific generic menu throughout ALL-IN-1 achieves consistency within the application's interface and between the numerous applications subsystems.

5.1 The Primary Menu Form

The primary menu form is the first menu which ALL-IN-1 displays to the user when they enter a particular subsystem. Following is a figure of the primary menu form and a description of the various fields.

```
User Name                User Title                Day Date
                          Subsystem Name
                          (Optional message field)

SEL Select      Category:  <field 1>
                  Name     :  <field 2>
C   Create      Number    :  <field 3>
E   Edit
D   Delete
P   Print
R   Read
I   Index

01 Option 1
02 Option 2

Enter selection and press RETURN
```

Figure 5-1 A Subsystem Menu

- User Name contains the user's full name from the User Profile.
- User Title contains the user's title from the User Profile.
- Day contains the day of the week.
- Date contains the current date.
- Subsystem Name contains the name of the current subsystem.
- Optional message field shows the number of outstanding messages, for example, the number of unread mail messages. The field remains blank when there are no outstanding messages.

- Category, Name, and Number are fields which represent the currently selected items for this subsystem and which make up the Current Item Block.

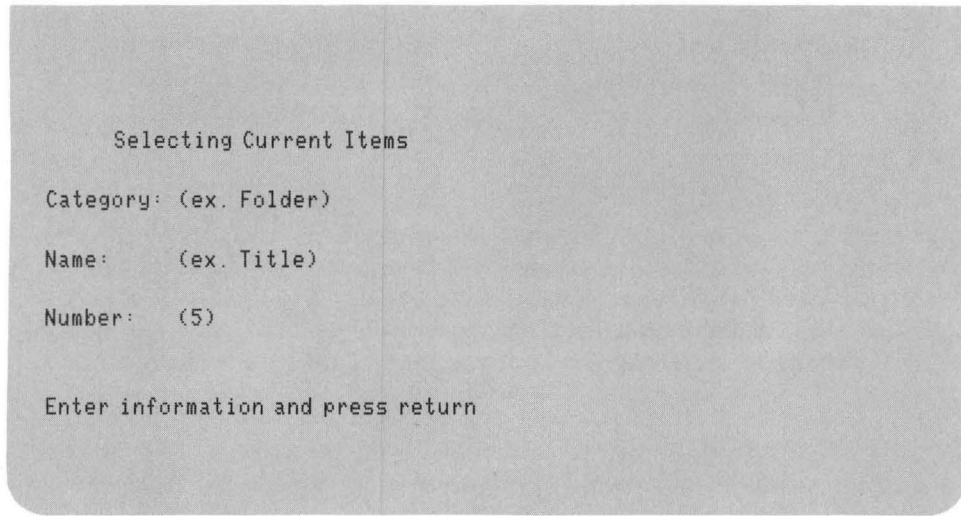
5.1.1 Current Items

An item is an entry you make. Some examples are: a document, a mail message, an action item, or a module. The current item fields display the selection attributes of the item you are currently working with or last worked with, in the particular subsystem. ALL-IN-1 maintains these items from one session to another, even if you leave ALL-IN-1 and the VMS system.

There are two, three, or four current item fields in a subsystem. The first field represents the most general organization of items, and the last field represents the most specific. Folder, Title, and Number are examples of current item fields in the Word and Document Processing subsystem. Library and Form are examples of current items in the Forms Development subsystem.

5.1.2 Select (SEL)

The Select option selects a new current item for processing, and the current item fields change to reflect the new selection. When Select is entered, an overlay form displays on the lower half of the screen. Thus the current items and the subsystem menu name remain on the screen so you can enter a new item while referring to the last item selected. An example follows. Often, the CHOICE keys are active to aid you in selecting these items.



```

Selecting Current Items

Category: (ex. Folder)

Name:      (ex. Title)

Number:    (5)

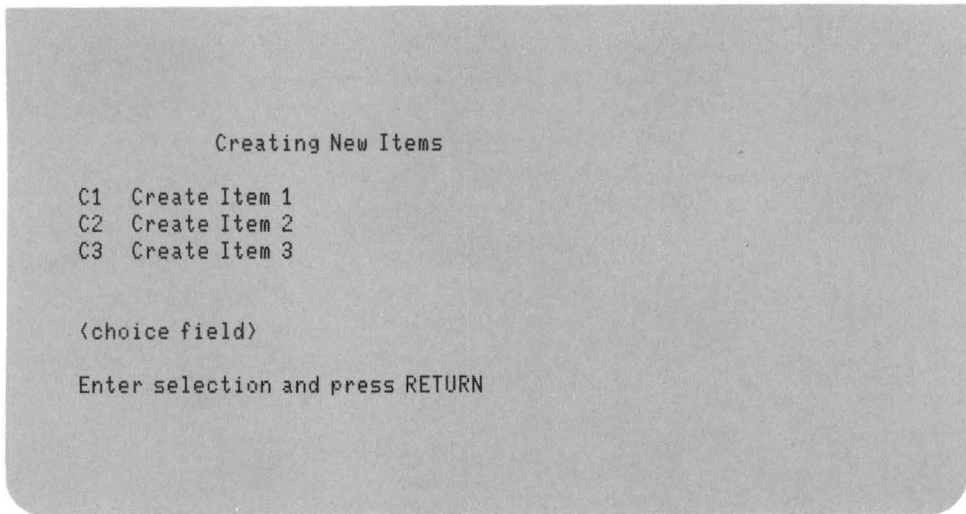
Enter information and press return

```

Figure 5-2 A Select Form

5.1.3 Create (C)

The Create option allows the user to create a new item in this subsystem. It may be a document, an action item, mail message, new FMS form, and so forth, depending upon the subsystem. This option, as in all of the generic options, displays either a menu screen or begins to process the user's request to create a new item. The menu forms that display should be overlay screens wherever possible. A title describing the menu screen is preferable.



```

                          Creating New Items

C1  Create Item 1
C2  Create Item 2
C3  Create Item 3

<choice field>

Enter selection and press RETURN

```

Figure 5-3 A Create Form

5.1.4 Edit (E)

The Edit option lets you modify an existing item. The Edit option may work differently depending on the subsystem, but the result is the same, modification of the current item.

5.1.5 Delete (D)

The Delete option removes the current item from a particular subsystem. ALL-IN-1 displays a confirmation message giving you the opportunity to cancel the operation. This is usually a one line prompt near the bottom of the screen. You respond Y and press RETURN to perform the deletion, or respond N and press RETURN to cancel it. You can also press EXIT SCREEN to cancel it.

5.1.6 Print (P)

The Print option lets you print an item to different destinations. It formats the output in the same order as it appears on your screen. In some cases, it is possible to print both a brief or a full description of the items in a subsystem.

The Print option displays the Printing Document form from which you can determine the output destination, the number of copies, and, by pressing **ADDITIONAL OPTIONS**, printer parameters (like the number of lines on a page, vertical spacing, margins, and shadow printing).

5.1.7 Read (R)

The Read option displays the current item on the terminal screen. Read either displays an item (as with a document or a mail message), or shows you information about the current item (as with an action item or a phone directory entry).

5.1.8 Index (I)

The Index option lets you view a list of items in the current subsystem. The items which appear in the list are based upon your selection criteria. These items are usually all contained within the highest level current item field.

Index also produces a file containing the list of items. This list is used with the **NEXT SELECTION** key to rapidly select these items.

5.1.9 Option x (Ox)

Option x represents options that are specific to a subsystem.

5.1.10 Training (TR)

The Training option lets you access the Computer Based Instruction (CBI) courses available for most subsystems. This option does not appear on subsystem menus.

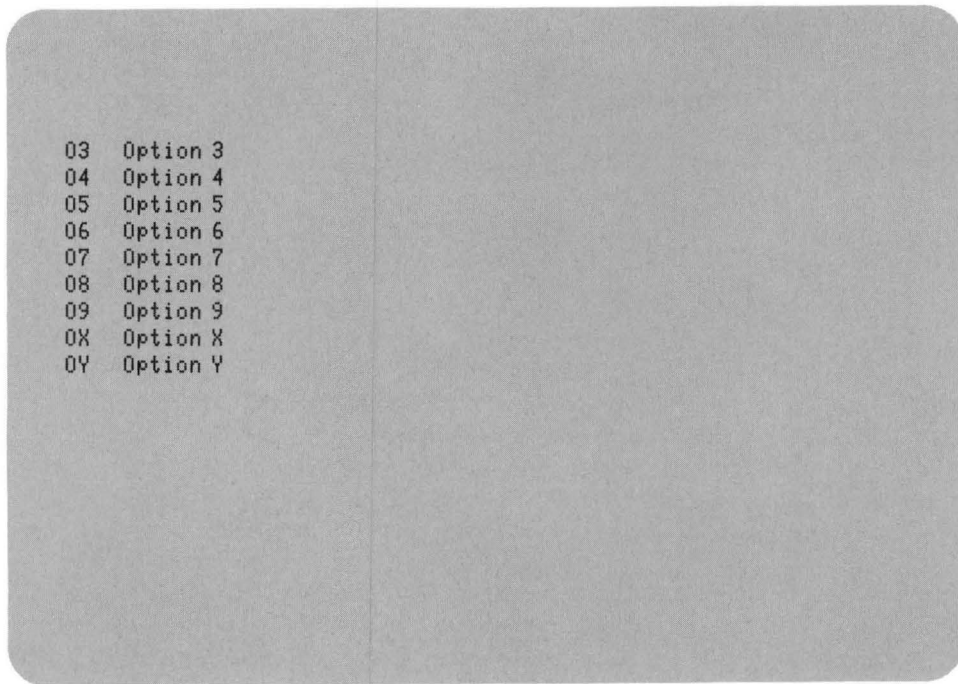
5.2 More Menu Options (M)

More Menu Options appears only on the main menu for the subsystem. When you select this option, an overlay form displays with additional options to the subsystem. Pressing Gold A performs the same function.

5.3 Secondary Menu Form — Additional Menu Options

The secondary menu displays options which are used less frequently in a subsystem. There may be more than one secondary menu. Selecting the More Menu Option (M) or pressing a Gold A key sequence (for additional topics) causes an overlay form to display this secondary menu on the screen. If there are no additional options, the message, No additional topics available, displays on line 24.

Together, the primary menu and the secondary menu make up a form-set. The secondary menu contains a choice field, identical to the choice field on the primary menu. Thus, the user can enter any option available within the subsystem on either menu. Following is an example of a secondary menu form.



A secondary menu form, represented as a gray rounded rectangle. It contains a list of options, each preceded by a two-digit code. The options are:

- 03 Option 3
- 04 Option 4
- 05 Option 5
- 06 Option 6
- 07 Option 7
- 08 Option 8
- 09 Option 9
- 0X Option X
- 0Y Option Y

Figure 5-4 A Secondary Menu Form

Index

This index references pages in all volumes: Volume 1 comprises the Preface through Chapter 5; Volume 2 comprises Chapter 6; Volume 3 comprises Chapter 7 through the Appendixes and the Glossary.

" and '

string delimiters

and /COMPUTE, 4-7

and /GET form

qualifier, 3-16

and /HARD, 3-16, 4-18

and /LOG, 3-19

and

/POST_FUNCTION, 3-21,
4-24

and

/PRE_FUNCTION, 3-21,
4-25

and /PROMPT, 4-26

and /SHOW, 4-45

and COMMAND function

parameters, 6-32

and CXP SEND, 12-8

and CXP WAIT, 12-9

and FOR function RSEs, 6-88

and GET, 6-107

and OA\$TABLE, C-8

and QUEUE_BATCH

qualifiers, 6-280

and QUEUE_PRINT

qualifiers, 6-285

in symbols, 8-2

in subprocess

See also ', 7-6

",'

string delimiters

in DSRs, 8-6

#

local-symbol prefix, 8-4

\$

and DCL function, 6-51

permanent-symbol prefix, 8-3

- &**
 - MERGE directive prefix, 6-138**
- ,**
 - See also* " and '
 - symbol-substitution character
 - in subprocess, 7-6
 - ' , " and string delimiters
 - and OA\$TABLE, C-8
- - continuation character
 - and COMPUTE, 6-41
 - in WRITES
 - to OAMAILBOX, 7-7
- .**
 - as field identifier in
 - FOR-function RSEs, 6-89
 - FORM-function substitution
 - character, 6-95
 - substitution character, 3-4
- /**
 - function-qualifier delimiter, 6-3
- :**
 - as a symbol-substring delimiter, 8-10
 - as an alternate-key delimiter
 - in DSRs, 8-6
 - Mail distribution list suffix, 11-3
- <**
 - function-call prefix, 3-32, 6-2
 - in a validation expression, 4-20, 4-30, 4-49, 4-54
 - pipeline source prefix, 6-9
- <->**
 - continuation characters
 - for MERGE template
 - directives, 6-139
- =**
 - buffer identifier in default
 - editor, 10-5
 - Help cross-reference prefix, 14-16
- @**
 - iterative symbol substitution
 - prefix, 8-12
- Mail distribution list prefix, 11-3
- Remote-node mail prefix, 11-4
- **
 - function-call delimiter
 - function list, 3-21
 - in subprocess, 7-7
 - in user-defined
 - functions, 6-11
- **
 - function call delimiter
 - in .IF expressions, 15-18
 - function-call delimiter
 - in FOR-function calls, 6-87
- - SPECIAL mail destination
 - prefix, 11-4
- A**
 - ACMSTASK function, 6-13 to 6-15
 - Action Items
 - checking
 - with AI__CHECK, 6-17 to 6-18
 - class, 3-42
 - entry form AIE
 - field information, 3-40
 - item, 3-42
 - priority, 3-42
 - schedule date, 6-17
 - status, 6-17
 - storage of, 3-40
 - type, 3-42, 6-17
 - Additional options
 - and Gold A, 4-56
 - ADE function, 6-16
 - AI__CHECK function, 6-17 to 6-18
 - ALL-IN-1
 - and the VAX Information
 - Architecture, 1-4
 - customizing
 - suggested prerequisites, xliii
 - exercises, A-1 to A-22

- facilities
 - primary, 1-7
 - flow control, 1-4
 - organization, 1-6
 - overview, 1-1 to 1-7
 - performance
 - See* Performance
 - subsystems
 - overview, 1-6
 - symbols, 8-1 to 8-36
- APPEND function, 6-19 to 6-20
- Application
 - debugging an
 - with Gold (, 6-34
 - defined, 1-6
 - design guidelines, 16-18
 - "standard", 16-19
 - subsystem versus, 1-6
- APR
 - conventions, 7
 - organization of, 2
 - overview, 1 to 8
 - related documents, 4, 5 to 7
 - using the, 3
- Architecture, VAX Information
 - overview, 1-3
- Argument form, 3-56 to 3-57
 - Named Data, 3-56
 - qualifiers, 3-57
 - .TYPE ARG, 3-5
- ATTACH function, 6-21 to 6-22
- Attributes, document, 15
- Attributes, FMS field
 - used to define a field, 4-2
- /AUTO field qualifier, 4-5
 - and /RECOGNITION, 4-28
 - and /RSE_RECOG, 4-31
- Autorecognition
 - with /AUTO, 4-5

B

- BACKSPACE
 - and OA\$FLD_PREV, 4-56

- Batch jobs
 - See* QUEUE_BATCH
 - /BEGIN form qualifier, 3-10, 3-27, 3-47
 - and /FIELDS, 3-49
 - in menus, 3-11
 - /BLANK field qualifier, 4-5 to 4-6
 - Boolean Expression, 4
 - Boolean expression, 6-86
 - in CABINET SCAN calls, 9-44
 - in FOR-function calls, 6-85
 - in search-form RSE, 3-60, 3-63
 - Boolean operators, 4
 - in .SFILE directive, 3-66
 - in FOR-function calls, 6-85
 - in search form RSE, 3-64
 - Brace character, left
 - used as script mnemonic delimiter, 15-3
 - Brace character, right
 - used as script mnemonic delimiter, 15-3
 - BYE function, 6-23 to 6-24

C

- CAB\$
 - File Cabinet DSR, 9-16
 - CAB\$ATTACH, 9-17
 - CAB\$ATTRIBUTES, 9-17
 - in a search form, 3-63
 - in a select form, 3-71
- CABINET function, 6-25, 9-20 to 9-47
 - subfunctions
 - See also* CABINET HELP
 - ADD_ATTRIBUTE, 9-21
 - ATTACH_DOCUMENT, 9-21
 - BEGIN, 9-22
 - CHANGE_ATTRIBUTE, 9-22
 - CLEAR_CURDOC, 9-23
 - CLOSE, 9-23
 - CONVERT, 9-23
 - COPY, 9-24
 - CREATE, 9-25

CABINET function,
 CROSSFILE, 9-26
 CURRENT, 9-27
 DECREMENT_COUNT,
 9-27
 DELETE_ATTRIBUTE, 9-27
 DELETE_DOCUMENT,
 9-28
 DELETE_OR_REFILE,
 9-28
 DETACH_DOCUMENT,
 9-29
 EDIT_DOCUMENT, 9-30
 END, 9-30
 EXCHANGE, 9-30
 FILE_ATTACHMENT, 9-30
 FORCE, 9-31
 GET_PENDING, 9-31
 HELP, 9-32
 JANITOR, 9-33
 NEXT_DOCUMENT, 9-33
 NEXT_FOLDER, 9-35
 NEXT_NUMBER, 9-37
 NEXT_SCAN, 9-38
 NEXT_TITLE, 9-40
 OPEN, 9-42
 POP, 9-42
 PUSH, 9-42
 PUT_PENDING, 9-42
 REFILE_DOCUMENT, 9-43
 REORG_DOCDB, 9-44
 RESET_DELETE, 9-44
 See also CABINET
 RESET_DELETE
 SCAN, 9-44
 SELECT, 9-45
 SHARE_DOCUMENT, 9-9,
 9-46
 STOP_SCAN, 9-47
 summary, 9-13
 UNSHARE_DOCUMENT,
 9-47
 Calculation form, 3-57 to 3-59
 behavior, 3-57
 Named Data, 3-57
 qualifiers, 3-58
 .TYPE CALC, 3-6
 .TYPE CLEAR, 3-6
 .TYPE DTR, 3-6
 CALENDAR function, 6-26
 current date/time, 13-2
 data sets, 13-1
 used by, 13-30
 files used by, 13-3
 subfunctions, 13-3 to 13-27
 CALENDAR
 ADD_ATTENDEE, 13-3
 CALENDAR CANCEL
 APPOINTMENT, 13-4
 CALENDAR CANCEL
 MEETING, 13-4
 CALENDAR CLEAR
 ATTENDEES, 13-5
 CALENDAR CONVERT
 ACTION_ITEMS, 13-5
 CALENDAR CONVERT
 APPOINTMENTS, 13-5
 CALENDAR CONVERT
 MEETINGS, 13-6
 CALENDAR DISPLAY
 ANSWER, 13-6
 CALENDAR DISPLAY
 APPOINTMENT, 13-7
 CALENDAR DISPLAY
 GRAPH, 13-8
 CALENDAR DISPLAY
 MEETING, 13-9
 CALENDAR DISPLAY
 MONTH, 13-9
 CALENDAR DISPLAY
 NEXT APPOINTMENT,
 13-10
 CALENDAR DISPLAY
 NEXT MEETING, 13-11
 CALENDAR DISPLAY
 NEXT REPLY, 13-11
 CALENDAR DISPLAY
 REPLY, 13-12
 CALENDAR INITIALIZE
 FILES, 13-13
 CALENDAR INITIALIZE
 MONTH, 13-13

CALENDAR MAIL__TO
 __ATTENDEES, 13-14
 CALENDAR
 MAKE__ATTENDEE__
 SYMBOLS, 13-14
 CALENDAR
 MAKE__MEETING__
 SYMBOLS, 13-15
 CALENDAR PRINT, 13-16
 CALENDAR PRINT
 ALL, 13-16
 CALENDAR PRINT
 CUSTOM, 13-17
 CALENDAR PRINT
 DAY__REMIND, 13-17
 CALENDAR PRINT
 DAY__SCHEDULE, 13-18
 CALENDAR PRINT
 MONTH, 13-18
 CALENDAR PRINT
 TODO, 13-19
 CALENDAR PRINT
 TWOCAL, 13-19
 CALENDAR PRINT
 WEEK__REMINDERS,
 13-20
 CALENDAR PRINT
 WEEK__SCHEDULE,
 13-20
 CALENDAR PURGE, 13-20.1
 CALENDAR
 REMOVE__ATTENDEES,
 13-20.1
 CALENDAR
 REPLY__TO__MEETING,
 13-21
 CALENDAR RESCHEDULE
 APPOINTMENT, 13-21
 CALENDAR RESCHEDULE
 MEETING, 13-22
 CALENDAR SCAN, 13-22
 CALENDAR SCHEDULE
 APPOINTMENT, 13-23
 CALENDAR SCHEDULE
 MEETING, 13-23
 CALENDAR SET
 ALTERNATE__USER,
 13-24
 CALENDAR SET
 DATE, 13-25
 CALENDAR SET
 NETWORK OFF, 13-26
 CALENDAR SET
 NETWORK ON, 13-26
 CALENDAR VERSION,
 13-27
 /CAPTIVE menu qualifier, 3-26
 CBI
 exit keys
 Gold X, CTRL/C, 4-59, 15-4
 Character, left Brace
 used as script mnemonic
 delimiter, 15-3
 Character, right Brace
 used as script mnemonic
 delimiter, 15-3
 CHECK__DISKQUOTA function,
 6-26.1 to 6-26.2
 /CHOICE menu qualifier, 3-25
 /CLEAR field qualifier, 4-6
 /CLEAR form qualifier, 3-11
 and /CHOICE, 3-11
 CLEAR function, 3-21, 6-27 to
 6-28
 versus /OVERLAY, 6-28
 CLOSE__PRIOR function, 6-29 to
 6-30
 in function lists, 6-7
 COMMAND function, 6-31 to 6-36,
 7-19
 and mailboxes, 7-2
 debugging tools used with, 6-34
 passing parameters with, 7-16
 simulating search strategy of
 with OA\$LIB:FINDCOM,
 6-35
 versus QUEUE__BATCH, 6-282
 Command procedure
 debugging
 with Gold (, OA\$LIB:V.COM,
 6-34

- Command procedures versus scripts performance in applications, 16-22
- Common Data Dictionary (CDD), 5
- Communications Control, 12-2 to 12-13
 - See also* CXP function
 - and the CXP function, 6-47
 - Control Documents, 12-2
 - customizing, 12-12
 - KEYPAD function, 6-117
 - Terminal Emulation Mode, 12-2
- /COMPUTE field qualifier, 4-7
- COMPUTE function, 6-37 to 6-41
 - from the subprocess, 7-10, 7-12
- Compute key
 - Gold #, 4-58
- Control Documents, 12-2
- CONTROL_C function, 6-42
- Converting to Version 2.X
 - form libraries, 16-9
- /COPY field qualifier, 3-79, 4-8 to 4-10
 - order of processing, 4-10
- COPY function, 6-43 to 6-44
 - as a pipeline source, 6-10
 - in Pipeline Mode, 6-44
- Copy Mode, hard, B-1 to B-5
 - See also* MODE function
 - providing information while in, 4-18
- CREATE function, 3-38, 6-45 to 6-46
- Customizing ALL-IN-1
 - performance considerations
 - See* Performance
 - with user-defined functions, 6-10 to 6-11
- Customizing documentation
 - for HELP, 16-24
 - for the user, 16-23
- Customizing forms, 16-8

- CXP function, 6-47, 12-2 to 12-13
 - See also* External Communications, Communications Control subsystem
 - subfunctions, 12-5
 - AT, 12-5
 - CONNECT, 12-5
 - CONNECT/DEVICE, 12-6
 - DISCONNECT, 12-6
 - DISPLAY, 12-6
 - PAUSE, 12-6
 - PROCESS, 12-7
 - RECORD, 12-7
 - restrictions on calling, 12-9
 - SEND, 12-8
 - SEND/DOCUMENT, 12-8
 - SET, 12-9
 - STOP RECORDING, 12-9
 - WAIT, 12-9

D

- /DATA entry form qualifier
 - and /KEY, 3-48
- Data file keys
 - ALL-IN-1 versus RMS, 3-39
 - specified with FDL, 3-39
- Data files
 - accessing
 - through alternate keys, 8-6
 - through DSRs, 8-4 to 8-9
 - with .FILE, 3-44
 - with /DATA, 3-44
 - adding unique key values to
 - with /INVALID, 4-21
 - converting
 - with FOR and WRITE, 6-91
 - copying
 - with COPY, 6-43 to 6-44
 - creating, 3-38, 16-4
 - with CREATE, 6-45 to 6-46
 - with OA\$INI_LOGICALS, 6-204
 - deleting, 6-56 to 6-57
 - modifying
 - with WRITE, 6-328 to 6-329

- updating existing records
 - with /VALID, 4-49
 - upgrading
 - with OA\$CNV__CONVERT, 6-161 to 6-162
 - with OA\$CNV__MERGE, 6-165 to 6-166
- Data set, 11
 - See also* DSR
 - text
 - See* Text data sets
- Data sets, C-1 to C-15
 - caching, 6-69, 6-169
 - CALENDAR, 13-1
 - data, C-1
 - See also* DSR
 - form, C-2
 - in a validation expression, 4-20, 4-29, 4-48, 4-51
 - in FOR function RSEs, 6-84
 - permanently open, 6-70
 - special
 - See* DSR
 - text, C-2
- Data validation
 - See* Validation
 - See also* DTR function
 - and DTR function, 6-66
 - and DTR\$ special symbols, 8-3
 - creating record definitions, 16-5
 - getting information from, 6-66
 - /DATE form qualifier, 3-12
 - Date/time formats, 6-49, 8-13
 - DATE__CONVERT function, 6-48 to 6-50
 - from the subprocess, 7-10
- Dates and times
 - formatting with
 - DATE__CONVERT, 6-48 to 6-50
- DCL
 - noninteractive commands
 - with OA\$DCL, 6-52
- DCL form
 - .TYPE DCL, 3-6
- DCL function, 6-51 to 6-53, 7-19, 16-15
 - and Gold \$, 4-58
 - and mailboxes, 7-2
 - called from the editor, 16-16
 - from the default editor, 10-17
- DCL key
 - Gold \$, 4-58
- DCL symbols
 - common, 7-12 to 7-15
 - and function lists, 7-15
 - RESULT, 7-12
 - TERMINATOR, 7-13
 - creating
 - with /COPY, 4-8, 7-9
 - with /PUT__SAVE, 4-27, 7-9
 - with COMPUTE, 6-38, 7-10
 - with DATE__CONVERT, 6-48, 7-10
 - with functions, 7-10 to 7-11
 - with GET, 6-102, 6-103, 7-11
 - with MAKE__FILE__NAME, 6-130, 7-11
 - with NEXT__LIST, 6-155
 - with NEXT__LIST, OA\$FLD__NEXT__LIST, 7-11
 - with PARSE__USER, 7-11
 - with SAVE, 6-293, 7-11
 - verifying assignment of, 16-15
- DCLMAILBOX
 - and OAMAILBOX, 7-4
 - mailbox to subprocess, 7-4
 - saving data that comes through
 - with RESULT, 7-12
 - with TERMINATOR, 7-13
- Debugging
 - applications
 - with Gold ((SET VERIFY), 6-34
 - command procedures
 - with Gold (, OA\$LIB:V.COM, 6-34
 - scripts
 - with Gold ((.TRACE ON), 6-34

- Debugging applications
 - with Gold (, 4-62
- Debugging features, 16-15
- /DECIMAL field qualifier, 4-10 to 4-11
 - versus DECIMAL function, 4-11
- DECIMAL function, 4-11, 6-54 to 6-55
- Default menu
 - form DEFAULT, 3-32, 4-56
 - keypad definitions in, 3-32
 - setting
 - with SET__MENU, 6-300 to 6-301
- Defining/redefining the keyboard, 4-63 to 4-65
- DELETE__FILE function, 6-56 to 6-57
- Desk Calculator form, 3-58 to 3-59, 4-60
 - formula mode, 3-59
 - mathematical functions, 3-59
 - setting decimal places, 3-59
 - .TYPE DESK, 3-6
 - using memory registers, 3-59
- DESTINATION function, 6-58, 10-10
 - and the Editor menu, 6-75
- DIGITAL Command Language
 - See* DCL
- Directives, keyboard
 - recognized at prompts, 6-269
- /DISPLAY form qualifier, 3-12
- DISPLAY function, 6-59 to 6-62
- Displaying information
 - through a scrolled region, 4-33
 - with /SHOW, 4-45
- Dispositions, form
 - terminator values, 4-69
- DO function, 6-63 to 6-64
 - See also* SCRIPT
- DO key
 - RETURN or Gold F, 4-58

- /DOC field qualifier, 4-11 to 4-14
 - and MAKE__FILE__NAME, 6-133
- Document attributes, 15
- Document, File Cabinet
 - See also* File Cabinet, Text files
 - attachments, 9-3, 9-21
 - attributes, 9-3, 9-8, 9-9, 9-13
 - See also* PDAF, SDAF
 - adding, 9-21
 - changing, 9-22
 - deleting, 9-27
 - compound, 9-3
 - detaching from, 9-29
 - filing attachments, 9-30
 - copying, 9-24
 - See also* Text files, copying
 - creating, 9-25
 - cross-filing, 9-26
 - current, 9-27
 - See also* OA\$CURDOC
 - deleting, 9-28, 9-33
 - See also* File Cabinet Wastebasket and Text files, deleting
 - with File Cabinet janitor, 9-4
 - editing, 9-30
 - See also* Text files, editing
 - next
 - by folder, 9-35
 - by number, 9-37
 - by title, 9-40
 - in a scan, 9-38
 - in folder, 9-33
 - number, 9-2
 - ordering, 9-6
 - refiling, 9-43
 - removing from shared area, 9-47
 - selecting, 9-45
 - sharing, 9-3, 9-46
- Documentation, customizing
 - for HELP, 16-24
 - for the user, 16-23

Down Arrow
 and OA\$FLD__DOWN, 4-58
 calls OA\$SCL__DOWN in
 scrolled region, 4-34
 DSR, 8-4 to 8-9
 alternate keys with, 8-6
 and /DATA, 3-48
 combining and nesting, 8-6
 File Cabinet, 9-16
 See also CAB\$
 in a validation expression,
 4-51
 with /INVALID, 4-20
 with /RECOGNITION, 4-29
 with /VALID, 4-48
 obtaining data with, 8-8
 scrolling
 SCROLL\$FUNCTION, 4-36
 to 4-42
 special, C-2
 OA\$DIR, C-3
 special field names
 %ADDRESS, 8-7
 %COUNT, 8-7
 %KEY, 8-8
 %NEXT, 8-8
 %WHOLE, 8-8
 syntax, 8-4
 validation
 OA\$DATE, C-2
 OA\$TABLE, C-8
 OA\$YESNO, C-9
 OA\$YN, C-9
 DTR function, 6-65 to 6-68
 DUMP__CACHE function, 6-69 to
 6-70, 16-11
 and OA\$FLO__CLOSE__LIB,
 6-194
 and OA\$FLO__OPEN__LIB,
 6-196

E

Edit form
 .TYPE EDIT, 3-6

EDIT function, 6-71 to 6-73
 Editing (default editor), 10-4 to
 10-19
 See also EDIT function
 Change Mode, 10-4
 commands
 nokeypad commands
 switching to Line Mode
 from, 10-18
 creating new files from
 editor, 10-17
 edit buffers, 10-5
 including text into
 document, 10-16 to 10-17
 initialization file, 10-8
 initializing, 10-4
 key numbers, 10-12
 Line Mode, 10-4
 commands, 10-4
 APPENDSR, 10-15
 BUFFER, 10-5
 CHANGE, 10-8, 10-9, 10-19
 CLEAR, 10-9
 COPY, 10-15
 CUTSR, 10-14
 DEFINE KEY, 10-6
 DEFINE MACRO, 10-5
 FIND, 10-9
 INCLUDE, 10-16
 INSERT, 10-9
 PASTE, 10-14
 RESET, 10-14
 SEL, 10-14
 SHOW KEY, 10-11
 WRITE, 10-17
 macros, 10-5
 invoking with key sequences,
 10-8
 moving text within a
 document, 10-14 to 10-16
 moving within a document,
 10-13
 nokeypad commands, 10-5
 ADV, 10-13
 and DEFINE KEY, 10-6
 BACK, 10-13

- Editing (default editor)
 - no keypad commands (Cont.)
 - BR, 10-14
 - C, 10-13
 - DEFK, 10-6
 - ER, 10-13
 - EXIT, 10-19
 - EXTEND (EXT), 10-8
 - L, EL, 10-13, 10-14
 - V, 10-13, 10-14
 - W, 10-13
 - XLATE, 10-10
 - and DESTINATION, 10-10
 - Editing (WPS-PLUS)
 - converting files, 10-21
 - Editor menu
 - and EDT function, 6-74 to 6-75
 - and function-call output, 6-75
 - EDT function, 6-74 to 6-75
 - Electronic Messaging, 11-1 to 11-50
 - See also* MAIL function
 - alternative MTS, 11-5
 - current message, 11-1
 - delivery mechanism, 11-3
 - distribution lists, 11-3
 - SUBSCRIBERS:, 11-4
 - header and text, 11-2
 - logicals
 - OA\$MAIL_NUM_ADDRESSES, 11-9
 - mail destinations, 11-4
 - menu interface, 11-2
 - nicknames, 11-3
 - special addressees
 - and SPECIAL.CMU, 11-6
 - and SPECIAL.COM, 11-6
 - @dest @node, 11-4
 - dist_list:, 11-4
 - I or ME, 11-3
 - PAPER MAIL, 11-3
 - __special-addressee, 11-4
 - special folders, 11-2
 - unrecognized destinations, 11-6
 - Entry form, 3-37 to 3-56
 - and data sets, 4-51
 - dynamically redefining, 3-78
 - key field recognition
 - and /SHOW, 4-46
 - Named Data, 3-37
 - processing, 3-54 to 3-56
 - required features, 3-37
 - .TYPE ENTRY, 3-6
 - using /STORE with, 4-46
 - ERROR function, 6-76 to 6-77
 - in function lists, 6-6
 - Errors
 - logging
 - with LOG, 6-124 to 6-125
 - trapping in function lists with
 - ERROR, 6-6, 6-76 to 6-77
 - EXIT function, 6-78 to 6-79
 - EXIT SCREEN
 - and the /POST_FUNCTION
 - field qualifier, 4-24
 - and the prior form, 3-75
 - from the editor menu, 6-315
 - keypad 0
 - and OA\$FLD_EXIT, 4-59
 - trapping with IFEXIT, 6-6
 - EXIT_SCREEN, 3-20
 - Expression, boolean
 - in search-form RSE, 3-63
 - Expression, record selection
 - See* RSE
 - External Communications, 12-1 to 12-13
 - See also* CXP function,
 - Multinode features,
 - Communications Control subsystem
- F**
- Facility
 - defined, 1-6
 - FDL file specification
 - in entry form Named Data, 3-39
 - FHELP function, 6-80

Field

- as a symbol, 8-2
- assigning a permanent value to
 - with /BLANK, 4-5
- assigning values dynamically
 - with PUT_FIELD, 6-277 to 6-278
- attributes
 - assigned with FMS, 4-2
 - FMS versus ALL-IN-1, 4-3
- clearing
 - with /CLEAR, 4-6
- defined, 4-1
- defining
 - with field qualifiers, 4-2
 - with FMS field attributes, 4-2
- display-only, 3-13
- getting values from
 - with /PUT_SAVE, 4-27
- in a DSR, 8-4
- layout
 - viewing with Gold V, 4-63
- moving from one to another, 4-66
- postprocessing from
 - with /POST_FUNCTION, 4-24
- preparing for mathematical output
 - with /DECIMAL, 4-10
- preprocessing from
 - with /PRE_FUNCTION, 4-25
- prompting through
 - with /PROMPT, 4-26
- putting file names into
 - with /DOC, 4-11
- putting values into
 - with /COPY, 4-8
 - with /GET_SAVE, 4-16
- suppressing input
 - with /BLANK, 4-5
- validating contents of
 - with /FILE_CHECK, 4-14

- validation, 4-51 to 4-56
 - See also* Validation
 - ensuring a unique value
 - with /INVALID, 4-19
 - /VALID, 4-47
 - viewing valid values for
 - with /RECOGNITION, 4-28
- FIELD function, 4-2, 6-81 to 6-82
 - and /FIELDS form qualifier, 4-2, 6-81
 - and OA\$FLD_nnn
 - functions, 4-67
 - Field processing, 4-1 to 4-70
 - summary, 4-69 to 4-70
- Field qualifiers
 - /AUTO, 4-5
 - /BLANK, 4-5 to 4-6
 - /CLEAR, 4-6
 - /COMPUTE, 4-7
 - /COPY, 3-79, 4-8 to 4-10
 - /DECIMAL, 4-10 to 4-11
 - /DOC, 4-11 to 4-14
 - /FILE_CHECK, 4-14 to 4-16
 - /GET_SAVE, 4-16 to 4-18
 - /HARD, 4-18 to 4-19
 - /INVALID, 4-19 to 4-22
 - /LIST, 4-22 to 4-23
 - /OPTIONAL, 4-23 to 4-24
 - /POST_FUNCTION, 4-24 to 4-25
 - /PRE_FUNCTION, 4-25 to 4-26
 - /PROMPT, 4-26 to 4-27
 - /PUT_SAVE, 4-27 to 4-28
 - /RECOGNITION, 4-28 to 4-30
 - /RSE_INVALID, 4-30 to 4-32
 - /RSE_RECOG, 4-30 to 4-32
 - /RSE_VALID, 4-30 to 4-32
 - /SCROLL, 4-33 to 4-44
 - /SHOW, 4-45 to 4-46
 - /STORE, 4-46
 - types of
 - in-field, 4-4
 - postprocessing, 4-4
 - preprocessing, 4-4
 - validation, 4-4

- Field qualifiers (Cont.)
 - /UNIQUE, 4-47
 - /USE__FORM, 4-47
 - used to define a field, 4-2
 - /VALID, 4-47 to 4-50
- /FIELDS form qualifier, 3-13, 3-34
 - and /BEGIN, 3-13, 3-49
 - and /GET, 3-13
 - and /KEY, 3-49
 - and FIELD function, 6-81
 - in FORM function calls, 3-9
 - in menus, 3-26
- File Cabinet, 9-1 to 9-47
 - accessing data, 9-15 to 9-18
 - See also* Special symbols and CAB\$ data set
 - as a data set, 9-15
 - attachments, 9-3
 - See also* Document attachments
 - converting, 9-23
 - Document, 9-2
 - See also* Document, Text files attached
 - See* Document attachments compound
 - See* Document, compound number, 9-2
 - sharing, 9-3
 - folder, 9-3
 - janitor, 9-4
 - organization, 9-4 to 9-8, 9-13
 - See also* individual file names
 - personal area, 9-4
 - reorganizing, 9-44
 - searching the
 - with search forms, 3-63
 - with select forms, 3-71
 - shared area, 9-19
 - switching
 - with NEWDIR, 6-150
 - using the, 9-13 to 9-19
 - Wastebasket
 - and CABINET JANITOR, 9-33
- File Cabinet special symbols
 - See* Special symbols, 9-18
- File Definition Language
 - See* FDL
- .FILE directive
 - in entry form Named Data, 3-38
- File names
 - creating
 - with /DOC, 4-11
- /FILE__CHECK field qualifier, 4-4, 4-14 to 4-16
- Files
 - See* Text files, Data files
 - copying with COPY, 6-43 to 6-44
 - deleting
 - with DELETE__FILE, 6-56 to 6-57
 - managing
 - with the File Cabinet, 9-1 to 9-47
 - naming
 - with MAKE__FILE__NAME, 6-130 to 6-133
 - processing, 10-32
 - purging
 - with PURGE__FILE, 6-273 to 6-276
- Flow control, 2-1 to 2-6
 - ALL-IN-1, 1-4
 - between ALL-IN-1 and its subprocess, 7-3
 - framework, 2-4 to 2-6
- FMS field attributes
 - used to define a field, 4-2
- FMS field highlights
 - performance in applications, 16-23
- FMS forms
 - ALL-IN-1 use of, 3-1
- FOR function, 6-83 to 6-91
 - and /RSE__nnn field qualifiers, 4-32
 - and validation qualifiers, 6-90

- and WRITE
 - converting files, 6-91
 - called, 4-31
 - data sets in, 6-84
 - DO subfunctions
 - in select forms, 3-72
 - in search forms, 6-90
 - in select forms, 3-69, 6-90
 - RSEs in, 6-83
 - selecting unique records
 - with, 6-84
- FORCE function, 6-92 to 6-93
 - in function lists, 6-7, 6-92
- Form
 - caching, 6-69, 6-169
 - called directly from menu
 - choice field, 3-31
 - creating, 16-4
 - customizing, 16-8
 - displaying
 - with SHOW__FORM, 6-302
 - dispositions
 - and OA\$FORM__DISPOSE, 4-66
 - DONE, 4-68
 - EXIT, 4-68
 - STAY, 4-68
 - summary of, 4-68
 - dynamically redefining, 3-8
 - maintaining, 16-2 to 16-12
 - memory-resident, 16-10
 - modifying, 16-5
 - multipage
 - See Form set
 - nonstandard, 3-80
 - redefining
 - with FORM function calls, 3-75 to 3-78
 - saving image of
 - with PRINT__SCREEN, 6-265
 - segmented
 - See Form set
 - testing, 16-11
- Form (Cont.)
 - .TYPE
 - ARG, 3-5
 - CALC, 3-6
 - CLEAR, 3-6
 - DCL (special), 3-6
 - DESK (special), 3-6
 - DTR, 3-6
 - EDIT (special), 3-6
 - ENTRY, 3-6
 - HELP (special), 3-6
 - in FORM function calls, 3-76
 - LIST (special), 3-6
 - MENU, 3-6
 - Named Data entry, 3-5
 - SEARCH, 3-6
 - SELECT, 3-6
 - VIEW, 3-6
 - Form and function
 - in ALL-IN-1, 1-5
 - in the VAX Information Architecture, 1-3
 - Form dispositions
 - terminator values, 4-69
 - FORM function, 3-3, 6-94 to 6-96
 - and OA\$FLD__nnn functions, 4-67
 - form .TYPE entry in calls to, 3-76
 - in function lists, 6-7
 - redefining Named Data with, 3-75
 - substitution character with, 3-4
 - Form Libraries, 3-2, 16-9
 - closing
 - with
 - OA\$FLO__CLOSE__LIB, 6-193 to 6-194
 - interactively updating, 16-17
 - opening
 - with
 - OA\$FLO__OPEN__LIB, 6-195 to 6-196
 - opening/reopening
 - with NEWLIB, 6-152 to 6-154

Form Libraries (Cont.)

- search order, 3-3
- searching, 16-9
- status of
 - and
OA\$FBT__LIST__LIBTREE,
6-175
 - and OA\$FBT__LIST__TREE,
6-177

Form processing, 3-1 to 3-80

- See also* individual form
- .TYPES

Form qualifiers, 3-8

- in function calls, 3-8
- common, 3-10 to 3-24
 - /BEGIN, 3-10
 - /CLEAR, 3-11
 - invalid with entry forms,
3-11
 - /DATE, 3-12
 - /DISPLAY, 3-12
 - /FIELDS, 3-13
 - /GET, 3-14
 - /HARD, 3-16
 - /HELP, 3-17
 - /HIGHLIGHT, 3-17
 - /KEYPAD, 3-18
 - /LOG, 3-19
 - /MAIL, 3-19
 - /NEXT, 3-19
 - /OVERLAY, 3-20
 - /POST_FUNCTION, 3-21
 - /PRE_FUNCTION, 3-21
 - /RESET, 3-22
 - /SUPER, 3-23
 - /TITLE, 3-23
 - /USER, 3-24
- entry form
 - /BEGIN, 3-47
 - /DATA, 3-48
 - /FIELDS, 3-49
 - /KEY, 3-50
 - /MODE (required), 3-44
 - /NOWRITE, 3-52
 - /ONCE, 3-52
 - /ONE_ENTRY, 3-53

Form qualifiers (Cont.)

- /SAVE__START, 3-46, 3-53
- in function calls
 - rules for processing, 3-8
- menu
 - /CAPTIVE, 3-26
 - /CHOICE (required), 3-25
 - /FIELDS, 3-26
 - /MORE, 3-27
 - /ONCE, 3-28
- select form
 - /LIST, 3-69
 - /STYLE, 3-70
- Form set, 3-78 to 3-80, 20
 - duplication of field names in,
3-79
 - entering first field in
 - with Gold T, 4-62
 - entering last field in
 - with Gold B, 4-57
 - entering next page in
 - with Gold TAB, NEXT
SCREEN, 3-79, 4-60
 - entering previous page in
 - with Gold BACKSPACE,
PREV SCREEN, 4-61
 - Gold key functions for, 3-80
 - restrictions on creating, 3-79
 - .TYPE directive in, 3-78
- .FORM__SET directive
 - See also* Form set
 - used to create a form set, 3-78
- FORMAT entry form, 10-29
- FORMAT function, 6-97 to 6-100
 - as a pipeline source, 6-10
- Formatting files
 - See* Text files, formatting
- Forms, customizing, 16-8
- Forms, precompiling
 - with the PFC option, 16-11
- Function lists
 - creating loops in
 - with REPEAT, 6-7
 - ERROR in, 6-6
 - forcing displays in
 - with FORCE, 6-7

from the subprocess, 6-8, 7-7
and common DCL symbols,
7-15

IFEXIT in, 6-6

trapping error conditions in,
6-6

with /POST__FUNCTION
qualifiers, 3-21, 4-24

with /PRE__FUNCTION
qualifiers, 3-22, 4-25

Functions

See also individual functions

ACMSTASK, 6-13 to 6-15

ADE, 6-16

AI__CHECK, 6-17 to 6-18

APPEND, 6-19 to 6-20

ATTACH, 6-21 to 6-22

BYE, 6-23 to 6-24

called from validation
expression, 4-20, 4-30, 4-49,
4-54

called through keyboard
directives, 4-63

calling, 6-1

CHECK__DISKQUOTA, 6-26.1
to 6-26.2

CLEAR, 6-27 to 6-28

CLOSE__PRIOR, 6-29 to 6-30

COMMAND, 6-31 to 6-36

COMPUTE, 6-37 to 6-41

CONTROL__C, 6-42

COPY, 6-43 to 6-44

CREATE, 6-45 to 6-46

CXP, 6-47

DATE__CONVERT, 6-48 to
6-50

DCL, 6-51 to 6-53

DECIMAL, 6-54 to 6-55

DELETE__FILE, 6-56 to 6-57

DESTINATION, 6-58

DISPLAY, 6-59 to 6-62

DO, 6-63 to 6-64

DTR, 6-65 to 6-68

DUMP__CACHE, 6-69 to 6-70

EDIT, 6-71 to 6-73

Functions (Cont.)

EDT, 6-74 to 6-75

ERROR, 6-76 to 6-77

EXIT, 6-78 to 6-79

FHELP, 6-80

FIELD, 6-81 to 6-82

FOR, 6-83 to 6-91

FORCE, 6-92 to 6-93

FORM, 6-94 to 6-96

FORMAT, 6-97 to 6-100

general syntax, 6-2

GET, 6-101 to 6-107

HELP, 6-108 to 6-111

IFEXIT, 6-112 to 6-113

INCLUDE, 6-114 to 6-115

interactive

and terminator sequences,
7-13

internal terminator, 4-66

KEYPAD, 6-116 to 6-117

LIGHTS, 6-118 to 6-119

LIST, 6-120 to 6-123

listing the

with FHELP, 6-80

LOG, 6-124 to 6-125

LOGICAL, 6-126 to 6-128

MAKE__FILE__NAME, 6-130
to 6-133

MERGE, 6-134 to 6-140

MERGE__LINE, 6-141 to 6-142

MERGE__LIST, 6-143 to 6-146

MODE, 6-147 to 6-149

nesting

See FOR, MERGE, SCRIPT

NEWDIR, 6-150 to 6-151

NEWLIB, 6-152 to 6-154

NEXT__LIST, 6-155 to 6-156

OA\$CLI__GET__SYMBOL,
6-157 to 6-158

OA\$CLI__GET__VALUE, 6-159
to 6-160

OA\$CNV__CONVERT, 6-161
to 6-162

OA\$CNV__DIST__LIST, 6-163
to 6-164

Functions (Cont.)

OA\$CNV_FROM_SCROLL,
6-167, 6-224 to 6-227
OA\$CNV_MERGE, 6-165 to
6-166
OA\$CNV_TO_SCROLL,
6-168, 6-224 to 6-227
OA\$DSA_CACHE_STATUS,
6-169 to 6-170
OA\$ENT_SET_xxx, 6-171
to 6-172
OA\$FBT_INSTALL_LIBRARY,
6-173 to 6-174
OA\$FBT_LIST_LIBTREE,
6-175 to 6-176
OA\$FBT_LIST_TREE, 6-177
to 6-178
OA\$FBT_REMOVE_LIBRARY,
6-179 to 6-180
OA\$FBT_WRITE_LIBRARY,
6-181 to 6-182
OA\$FLD_NEXT_LIST, 6-188
OA\$FLD_nnn, 6-183 to 6-187
See also individual functions
OA\$FLD_RECOGNITION,
6-189
OA\$FLD_SCROLL_OFF,
6-190
OA\$FLD_SEARCH, 6-191
OA\$FLO_CLEAR_VA, 6-192
OA\$FLO_CLOSE_LIB, 6-193
to 6-194
OA\$FLO_OPEN_LIB, 6-195
to 6-196
OA\$FRM_SET_FIELD, 6-197
OA\$HAR_CHECK_DECCRT,
6-198 to 6-199
OA\$INI_GLOBALS, 6-201
OA\$INI_INITIALIZE, 6-202
to 6-203
OA\$INI_LOGICALS, 6-204
OA\$INI_OPEN_LIB, 6-205
OA\$INI_SET_DEFAULT_DIR,
6-206
OA\$INI_SUBPROCESS, 6-207
to 6-208

Functions (Cont.)

OA\$KEY_COMMAND, 6-209
OA\$LIST_xxx, 6-210 to 6-211
OA\$LOG_DISPLAY, 6-212
to 6-213
OA\$LOG_TO_FILE, 6-214
to 6-215
OA\$MENU_LEVEL_PUSH,
6-216 to 6-217
OA\$MENU_SHOW_STACK,
6-218 to 6-219
OA\$MRG_TTOUT, 6-220 to
6-221
OA\$MSG_PURGE, 6-222
OA\$MSG_TRAP, 6-223
OA\$SCL_nnn, 6-224 to 6-227
See also individual functions
OA\$SCP_STACK_DUMP,
6-228
OA\$STAT_GET_STATISTICS,
6-229 to 6-230
OA\$SUB_CLOSE, 6-231
OA\$SUB_OPEN, 6-232
OA\$SYM_CLOSE, 6-233
OA\$SYM_GET_SYMBOL,
6-234 to 6-235
OA\$SYM_MAKE_SYMBOL,
6-236 to 6-237
OA\$SYM_OPEN, 6-238 to
6-239
OA\$TRA_OUT, 6-240
OA\$TRA_SET, 6-241 to 6-243
OA\$TRM_BRDCST_INIT,
6-244 to 6-244.1
OA\$TRM_BRDCST_STOP,
6-244.2 to 6-244.3
OA\$TRM_INIT, 6-245
OA\$TRM_INQUIRE, 6-246
OA\$TRM_PORT_LINE, 6-247
OA\$TRM_PORT_LOG_ON,
6-248
OA\$TRM_PORT_OFF, 6-249
OA\$TRM_PORT_ON, 6-250
OA\$TRM_PORT_PRINT,
6-251

Functions (Cont.)

OA\$TXL__CLOSE, 6-252
OA\$TXL__COMPILE, 6-253
OA\$TXL__DIRECTORY, 6-254
OA\$TXL__INSTALL, 6-255
OA\$TXL__LIST, 6-256
OA\$TXL__OPEN, 6-257 to 6-258
OA\$TXL__REMOVE, 6-259
OA\$VAL__SET__VALID, 6-260 to 6-261
PARSE__USER, 6-262 to 6-263
 pipelining, 6-9
 source function, 6-9
PORT__OFF, 6-264
PORT__ON, 6-264
PRINT__SCREEN, 6-265 to 6-266
PROMPT, 6-267 to 6-270
PROMPT__ID, 6-271 to 6-272
PURGE__FILE, 6-273 to 6-276
PUT__FIELD, 6-277 to 6-278
QUEUE__BATCH, 6-279 to 6-283
QUEUE__PRINT, 6-284 to 6-288
RENAME, 6-289 to 6-290
REPEAT, 6-291 to 6-292
SAVE, 6-293 to 6-295
SCRIPT, 6-296 to 6-297
SET__DEFAULT, 6-298 to 6-299
SET__MENU, 6-300 to 6-301
SHOW__FORM, 6-302
SORT__LIST, 6-303 to 6-305
SPAWN, 6-306 to 6-307
 stacking, 6-4 to 6-9
 See also Function lists
STATISTICS, 6-308 to 6-309
TEST__SPEC, 6-310 to 6-311
TIME, 6-312 to 6-313
 trapping errors from, 6-6
 types of, 6-1
UNWIND, 6-314 to 6-315
 user-defined, 6-10 to 6-11

Functions (Cont.)

 restriction on
 implementing, 6-10
 calling standards, 16-22
 VAX/VMS requirements,
 16-21
VERSION, 6-316 to 6-317
WAIT, 6-318 to 6-320
WPSPLUS\$CBI__CLOSE__
 WINDOW, 6-321
WPSPLUS\$CBI__CLOSE__
 WINDOW2, 6-322
WPSPLUS\$CBI__OPEN__
 WINDOW, 6-323 to 6-324
WPSPLUS\$CBI__OPEN__
 WINDOW2, 6-325 to 6-326
WPSPLUS__SPELL, 6-327
WRITE, 6-328 to 6-329
YESNO__PROMPT, 6-330 to 6-332
YESNO__PROMPT__ID, 6-333 to 6-334

G

Generic menu design, 5-1 to 5-8
/GET form qualifier, 3-14
 and /GET__SAVE field
 qualifier, 3-15
 processed after
 /PRE__FUNCTION, 3-14
GET function, 6-101 to 6-107,
 16-17
 and complex DSRs, 6-107
 from the subprocess, 7-11, 7-12
GET OA\$DCL, 7-19
GET OA\$FUNCTION
 special GET destination
 and SAVE, 6-294
/GET__SAVE field qualifier, 4-16
 to 4-18
 See also /GET
 and /GET form qualifier, 3-16
Gold BACKSPACE
 previous page in a form set,
 3-80

Gold TAB

next page in a form set, 3-80

Gold-key sequences

and Recognition, 4-46

in a scrolled region, 4-34

in menu choice field, 3-31

predefined, 4-56 to 4-63

H

Hard Copy Mode, 3-16, B-1 to B-5

See also MODE function

providing information while

in, 4-18

/HARD field qualifier, 4-18 to 4-19

/HARD form qualifier, 3-16

Help

and Gold H, 4-59

facility, 14-1 to 14-22

See also HELP function

additional topics, 14-14

to 14-18

HELP forms, 14-5

keyboard-style help, 14-5

library, 14-6

cross-references, 14-14 to

14-18

line-1 (direct), 14-17

modifying, 14-18

module format (menu), 14-10

module format

(nonmenu), 14-6

restrictions, 14-20

switching to another, 14-21

Help form

.TYPE HELP, 3-6

/HELP form qualifier, 3-17, 14-2

HELP function, 6-108 to 6-111

backing off, 14-4

called after DISPLAY, 6-61

called from a prompt, 6-269

context checking, 14-2

and choice-field contents,

14-3

from a prompt, 14-3, 14-22

from the editor, 14-3

Help function

context checking (Cont.)

on a menu, 14-2

on a nonmenu, 14-2

with /HELP, 14-2

from the default editor, 14-5

syntax restrictions, 6-110

/HIGHLIGHT form qualifier, 3-17

performance impact, 3-18

used with FMS field

highlighting, 3-18

I

IFEXIT function, 6-112 to 6-113

in function lists, 6-6

INCLUDE function, 6-114 to

6-115, 10-16

Indexed data files

See Data files

Indexed files, RMS

defined by entry forms, 3-37

Initialization and flow control, 2-1

to 2-6

partial, 2-3

standard sequence of, 2-2

Input Processing facility, 3-31

Installation Verification

Procedure

See IVP

Interrupt menu

and Gold I, 4-60

/INVALID field qualifier, 4-4, 4-19

to 4-22

and /RECOGNITION, 4-21

on entry form key fields, 3-46

Iterative symbol substitution

in FOR-function calls, 6-90

K

/KEY entry form qualifier, 3-50

and /BEGIN, 3-51

and /DATA, 3-48, 3-50

and /FIELDS, 3-49, 3-51

and /SAVE_START, 3-52

highlighting by, 3-51

- Key of reference, RMS
 - in an FDL-created file, 3-39
- Key, NEXT LIST
 - and /LIST field qualifier, 4-60
 - keypad ., 4-60
- Keyboard directives
 - See also* individual directives
 - recognized at prompts, 6-269
 - where they are recognized, 4-63
- Keyboard keys
 - default definitions of, 4-56 to 4-63
 - redefined by /SCROLL, 4-34
- Keyboard, defining/redefining the, 4-63 to 4-65
- /KEYPAD form qualifier, 3-18
- KEYPAD function, 6-116 to 6-117
- Keypad keys
 - See also* Terminator key(s)
 - defining/redefining, 4-63 to 4-65
 - 0 — EXIT SCREEN key, 4-59, 6-113
 - . — NEXT LIST key, 4-60
 - reserved keys, 4-65
- /KEYS entry form qualifier
 - ignored during file creation, 3-38
- Keys, data file
 - ALL-IN-1 versus RMS, 3-39
 - specified with FDL, 3-39

L

- Layered products
 - ACMS, 6-13 to 6-15
 - DATATRIEVE, 6-65
- Left Brace character
 - used as script mnemonic delimiter, 15-3
- Libraries, form, 3-2, 16-9
 - opening/reopening
 - with NEWLIB, 6-152 to 6-154
- LIGHTS function, 6-118 to 6-119
- /LIST field qualifier, 4-22 to 4-23
 - and /LIST select form qualifier, 3-70

- List form
 - .TYPE LIST, 3-6
- LIST function, 6-120 to 6-123
 - as a pipeline source, 6-10
- LIST key
 - Gold L, 4-57
- List processing, 10-21 to 10-27
 - See also* MERGE,
 - MERGE_LIST
 - with MERGE, 6-134 to 6-140
 - with MERGE_LINE, 6-141 to 6-142
- WPS-PLUS
 - MERGE_LIST, 6-143 to 6-146
 - SORT_LIST, 6-303 to 6-305
 - TEST_SPEC, 6-310 to 6-311
- /LIST select form qualifier, 3-69
 - and /LIST field qualifier, 3-70
 - and SEL_STYLE
 - subfunction, 3-70
- Local symbol
 - table, 27
- Local symbols, 8-4
- /LOG form qualifier, 3-19
- LOG function, 6-124 to 6-125
- LOGICAL function, 6-126 to 6-128
 - from the subprocess, 7-12
- Logical name, 27
 - See also* Logicals
- Logicals, 7-16 to 7-18
 - main process
 - and LOGICAL function, 6-126 to 6-128
 - main process versus
 - subprocess, 7-17

M

- /MAIL form qualifier, 3-19
- MAIL function, 6-129
 - See also* Electronic Messaging
 - subfunctions, 11-10 to 11-51
 - ANSWER, 11-11
 - ATTACH, 11-13
 - AUTO_FORWARD, 11-13
 - AUTO_REPLY, 11-14

MAIL function (Cont.)

CANCEL, 11-15
CANCEL__AUTO__REPLY,
11-15
CC, 11-15
CLOSE__MESSAGE, 11-16
CREATE, 11-16.1
DEFAULT__MESSAGE,
11-19
DEFER, 11-20
DELETE, 11-20
DELIVERY__RECEIPT,
11-21
DETACH, 11-22
DIST__COPY, 11-22
DIST__CREATE, 11-23
DIST__DELETE, 11-23
DIST__EDIT, 11-24
DIST__LIST, 11-24
EDIT, 11-25
EXCHANGE__CURMES,
11-26
EXPAND__DIST__LIST,
11-26
FILE__ATTACHMENT,
11-27
FILE__BODY, 11-28
FILE__MESSAGE, 11-29
FORWARD, 11-30
FORWARDABLE, 11-32
FUNCTION, 11-32
GET, 11-33
INDEX, 11-33
MAIL__DOCUMENT, 11-34
NEXT, 11-36
NO__DELIVERY__RECEIPT,
11-37
NON__FORWARDABLE,
11-37
ORIGINAL, 11-37
OUTBOX, 11-38
POP__CURMES, 11-38
PRINT, 11-39
PRIORITY, 11-40
PUSH__CURMES, 11-41

MAIL function (Cont.)

READ, 11-42
READ__RECEIPT, 11-43
SEND, 11-44
SHOW, 11-46
STATUS, 11-46
SUBJECT, 11-47
TEXT, 11-48
TO, 11-48
VAXMAIL__CHECK, 11-49

Mailboxes, 7-3

and the subprocess, 7-2

use of in applications, 16-20

MAKE__FILE__NAME

function, 6-130 to 6-133

from the subprocess, 7-11

Mathematical calculations

operators and operands for,

6-38

with COMPUTE, 6-37 to 6-41

with the Desk Calculator

form, 3-58

Mathematical computations

setting decimal places for, 4-11,

6-54 to 6-55

with /COMPUTE field

qualifier, 4-7

with Gold #, 4-58

Menu form, 3-24 to 3-36

choice field processing, 3-29

creating a complex, 3-33 to

3-36

Named Data, 3-25

processing

flow control, 3-24

qualifiers

/CAPTIVE, 3-26

/CHOICE (required), 3-25

/FIELDS, 3-26

/MORE, 3-27

/ONCE, 3-28

required features, 3-24

selecting the current item, 4-58,

4-62

selecting the first item, 4-62

.TYPE MENU, 3-6

- Menu form processing, 3-28
 - complex menus, 3-33 to 3-36
- Menu levels, 3-36
- Menu options, stacking, 3-32
 - restrictions when, 3-33
- Menu stack, prior, 3-35
- Menu stacks, 3-36
- MERGE function, 6-134 to 6-140
 - as a pipeline source, 6-10
 - examples, 10-22 to 10-27
 - flow control, 10-25, 10-27
 - header file
 - and OA\$PAGE_LENGTH 6-136
 - header-file, 10-26
 - list file, 10-24
 - and OA\$MERGE_KEY, 6-136
 - template directives, 6-137 to 6-139
 - continuation lines, 6-139
- MERGE_LINE function, 6-141 to 6-142
 - template directives with, 6-141
- MERGE_LIST function, 6-143 to 6-146
 - See also* SORT_LIST, TEST_SPEC
- Message buffer
 - viewing
 - with Gold W, 4-60
- Message facility, 14-22 to 14-26
 - See also* OAMESS.MSG and Help, 14-26
 - message buffer, 14-24
 - viewing, 14-25
 - message severity, 14-24
 - message syntax, 14-23
- MESSAGES key
 - Gold W, 4-60
- Mnemonic delimiters
 - script, 15-3
- Mnemonic, scripts, 15-3
- /MODE entry form qualifier, 3-44
- MODE function, 6-147 to 6-149
- Modes
 - Command (Hard Copy)
 - providing information while in, 4-18
- .MORE, 3-7
- .MORE directive, 3-7
- /MORE menu qualifier, 3-26, 3-27
 - in FORM function calls, 3-9
 - sample application of, 3-34
- Multinode features
 - overview, 12-1
- Multipage form
 - See* Form set
- N**
- Named Data, 3-28, 30
 - dynamically redefining, 3-75
 - with /RESET, 3-77
 - modifying, 16-6
 - reorganizing, 16-9
 - searching for, 3-7
 - use of by ALL-IN-1, 3-4
 - viewing
 - with Gold N, 4-60
- Nesting functions
 - See* FOR, MERGE, SCRIPT
- NEWDIR function, 6-150 to 6-151
- NEWLIB function, 6-152 to 6-154, 16-11
- /NEXT form qualifier, 3-19
 - and the prior menu stack, 3-75
- NEXT LIST key
 - and /LIST field qualifier, 4-60
 - keypad ., 4-60
- NEXT PAGE key
 - Gold TAB, NEXT SCREEN, 4-60
- NEXT_LIST function, 6-155 to 6-156
 - from the subprocess, 7-11
- \$NEXT_LIST_ITEM
 - maintained with NEXT_LIST, OA\$FLD_NEXT_LIST, 7-11

Nonstandard form, 3-80

/NOWRITE entry form qualifier,
3-52

O

&OA

function-call prefix, 6-2, 6-137

OA

function-call prefix, 6-2, 7-5

OA\$BUILD:OASDF.BLI

and user-defined functions 6-11

OA\$BUILD:HLPSOURCE.HLB

Help-text sources, 14-18

OA\$CLI_GET_SYMBOL

function, 6-157 to 6-158

OA\$CLI_GET_VALUE

function, 6-159 to 6-160

OA\$CNV_CONVERT function,

6-161 to 6-162

OA\$CNV_DIST_LIST

function, 6-163 to 6-164

OA\$CNV_FROM_SCROLL

function, 6-167, 6-224 to 6-227

See also /SCROLL

OA\$CNV_MERGE function, 6-165

to 6-166

OA\$CNV_TO_SCROLL

function, 6-168, 6-224 to 6-227

See also /SCROLL

OA\$CURDOC

special symbol

and CABINET

CLEAR_CURDOC, 9-23

and CABINET COPY, 9-25

and CABINET CREATE,

9-26

and CABINET

CROSSFILE, 9-26

and CABINET CURRENT,

9-27

and CABINET

DELETE_OR_REFILE,

9-29

and CABINET

NEXT_FOLDER, 9-36

and CABINET

NEXT_NUMBER, 9-37

and CABINET

NEXT_TITLE, 9-40

and CABINET SELECT, 9-45

OA\$DATA

and LOGICAL function, 6-128

subprocess versus main

process, 7-17

OA\$DATA:ATTENDEE.DAT

system attendee file, 13-1

OA\$DATA:CALACCESS.DAT

CALENDAR access file, 13-1

OA\$DATA:DAF.DAT

See SDAF

OA\$DATA:FORMAT.DAT

Format master file, 10-28

OA\$DATA:MEETING.DAT

system meeting file, 13-1

OA\$DATA:PENDING.DAT, 9-12

and CABINET

GET_PENDING, 9-31

and CABINET

PUT_PENDING, 9-42

system Pending file, 9-2

OA\$DATA:username.A1CAL

the personal calendar file, 13-1

OA\$DATE

special symbol

and DATE_CONVERT, 6-48

validation DSR, C-2

OA\$DATE_FULL

special symbol

and DATE_CONVERT, 6-48

OA\$DATE_NBS

special symbol

and DATE_CONVERT, 6-48

OA\$DCL

special GET destination, 6-52,

16-23

OA\$DEFAULT_EXTENSION

special symbol, 6-131, 10-3

used to create file name, 4-12

OA\$DIR

field-name keywords, C-3

special DSR, C-3

OA\$DISPLAY
 special GET destination, 6-61
OA\$DSA__CACHE__STATUS
 function, 6-70, 6-169 to 6-170
OA\$EDIT__BUFFER
 text data set, 10-10
OA\$EDIT__FILE
 special symbol, 6-71
OA\$EDIT__INI__FILE
 special symbol, 6-72
OA\$EDIT__PURGE__COUNT
 special symbol, 6-72
OA\$EDITOR
 special symbol, 6-73
OA\$ENT__SET__xxx function,
 6-171 to 6-172
OA\$FBT__INSTALL__LIBRARY
 function, 6-173 to 6-174
OA\$FBT__LIST__LIBTREE
 function, 6-175 to 6-176, 6-194,
 6-196
OA\$FBT__LIST__TREE
 function, 6-177 to 6-178
OA\$FBT__REMOVE__LIBRARY
 function, 6-179 to 6-180
OA\$FBT__WRITE, 16-11
OA\$FBT__WRITE__LIBRARY
 function, 6-181 to 6-182
OA\$FIELD__TERM
 special symbol, 7-13
 and BACKSPACE, 4-57
 and Gold BACKSPACE,
 PREV SCREEN, 4-61
 and Gold TAB, NEXT
 SCREEN, 4-61
 and keypad 0, 4-59
 and PROMPT, 6-267, 6-271
 and RETURN, 4-58
 and TAB, 4-62
OA\$FLD__BOTTOM function,
 4-67, 6-183
 and Gold B, 4-57
OA\$FLD__DONE function,
 4-67, 6-184
 and PROMPT, 6-267, 6-271
 and RETURN, Gold F, 4-58
OA\$FLD__DOWN function, 4-67,
 6-184
 and Down Arrow, 4-58
OA\$FLD__EXIT function,
 4-67, 6-184
 and IFEXIT, 6-113
 and keypad 0, Gold K, Gold
 Q, 4-59
 and PROMPT, 6-267, 6-271
 and the prior form, 3-75
OA\$FLD__NEXT function, 6-184
 and TAB, 4-62
OA\$FLD__NEXT__LIST
 function, 4-67, 6-188
 and /LIST field qualifier, 4-60
 and the NEXT LIST key, 4-60
 from the subprocess, 7-11
OA\$FLD__nnn functions, 4-66,
 6-183 to 6-187
 See also individual functions
OA\$FLD__PAGE__BACK
 function, 4-67, 6-185
 and Gold BACKSPACE, PREV
 SCREEN, 4-61
OA\$FLD__PAGE__BOTTOM
 function, 6-185
OA\$FLD__PAGE__FORWARD
 function, 4-67, 6-185
 and Gold TAB, NEXT
 SCREEN, 4-60
OA\$FLD__PAGE__TOP
 function, 6-185
OA\$FLD__PREV function, 4-67,
 6-185
 and BACKSPACE, 4-56
OA\$FLD__RECOG function
 and Gold L, 4-57
OA\$FLD__RECOGNITION
 function, 6-189
OA\$FLD__SCROLL__OFF
 function, 6-190
OA\$FLD__SEARCH function,
 6-191
 and Gold S, 4-57

OA\$FLD__TOP function, 4-67
 and Gold T, 4-62
OA\$FLD__UP function, 4-67, 6-186
 and Up Arrow, 4-62
OA\$FLO__CLEAR__VA
 function, 6-192
OA\$FLO__CLOSE function
 and DUMP__CACHE, 6-70
OA\$FLO__CLOSE__LIB
 function, 6-193 to 6-194
 and DUMP__CACHE, 6-194
 and NEWLIB, 6-194
OA\$FLO__OPEN function,
 16-11
 and DUMP__CACHE, 6-70
OA\$FLO__OPEN__LIB function,
 6-195 to 6-196
 and DUMP__CACHE, 6-196
 and NEWLIB, 6-196
OA\$FORM__DISPOSE
 special symbol
 See also Form dispositions
 and CABINET, 6-6
 and OA\$FLD__BOTTOM,
 4-57
 and OA\$FLD__DONE, 4-58
 and OA\$FLD__EXIT, 4-59
 and OA\$FLD__NEXT, 4-62
 and OA\$FLD__PAGE__
 BACK, 4-61
 and OA\$FLD__PAGE__
 FORWARD, 4-61
 and OA\$FLD__PREV, 4-57
 and OA\$FLD__TOP, 4-62
 and YESNO__PROMPT,
 6-330, 6-334
 set by OA\$FLD__nnn
 functions, 4-67
 summary of values for, 4-68
OA\$FORM__TERMINATOR
 special symbol, 7-13
 and BACKSPACE, 4-57
 and Gold BACKSPACE,
 PREV SCREEN, 4-61
 and Gold TAB, NEXT
 SCREEN, 4-61
 and keypad 0, 4-59
 and PROMPT, 6-267, 6-271
 and RETURN, 4-58
 and TAB, 4-62
 and YESNO__PROMPT,
 6-330, 6-334
 terminator values, 4-69
OA\$FRM__SET__FIELD
 function, 6-197
OA\$FUNCTION
 special GET destination
 and ERROR, 6-77
 and LIST, 6-122
 and LOG, 6-125
 and LOGICAL, 6-128
 and MERGE, 6-140
 and NEWDIR, 6-151
 and PURGE__FILE, 6-275
OA\$FUNCTION, GET
 special GET destination
 and SAVE, 6-294
OA\$HAR__CHECK__DECCRT
 function, 6-198 to 6-199
OA\$HELP__BOTTOM
 HELP form, 14-5
OA\$HELP__TOP
 HELP form, 14-5
OA\$INI__GLOBALS function,
 6-201
OA\$INI__INITIALIZE function,
 6-202 to 6-203
OA\$INI__LOGICALS function,
 6-204
OA\$INI__OPEN__LIB function,
 6-205
OA\$INI__SET__DEFAULT__DIR
 function, 6-206

OA\$INI__SUBPROCESS
 function, 6-207 to 6-208
OA\$KEY__COMMAND
 function, 6-209
OA\$LIB
 and LOGICAL function, 6-128
 subprocess versus main
 process, 7-17
OA\$LIB:ACTITEM.FDL, 3-42
 used to create
 OAUSER:ACTITEM.DAT, 3-42
OA\$LIB:FINDCOM.COM
 and SET__DEFAULT, 6-299
OA\$LIB:MEMRES.FLB
 memory-resident form
 library, 3-2, 16-10
OA\$LIB:OAFORM.FLB
 altering search order of, 3-3
 common form library, 3-2
OA\$LIB:OAHELP.HLB
 See also Help library
 ALL-IN-1 help library, 6-110
 default help library, 14-7
OA\$LIB:STANDARD.FLB
 prototype form library, 3-2
OA\$LIB:USER.FLB
 user's form library, 3-2
OA\$LIST__xxx function, 6-210
 to 6-211
OA\$LOG__DISPLAY function,
 6-212 to 6-213
OA\$LOG__TO__FILE function,
 6-214 to 6-215
OA\$MAIDES
 validation DSR, C-7
OA\$MAIL__ADD__ADDR
 special validation DSR, 11-50
OA\$MAIL__ADDRESS
 special validation DSR, 11-50
 validation DSR, C-8
OA\$MAIL__COUNT
 special symbol
 and CABINET
 GET__PENDING, 9-32
OA\$MAIL__NUM__ADDRESSES
 MAIL-related logical, 11-9
OA\$MAILPRIO
 special validation DSR, 11-50
 validation DSR, C-8
OA\$MENU__LEVEL__PUSH
 function, 6-216 to 6-217
OA\$MENU__REMAINDER
 special GET destination, 3-33
OA\$MENU__SHOW__STACK
 function, 6-218 to 6-219
OA\$MERGE__KEY
 special symbol
 and MERGE, 6-136
 and MERGE list file, 10-25
OA\$MERGE__LINE
 special GET destination, 10-26
 and MERGE__LINE, 6-142
OA\$MERGE__REC
 See OA\$MERGE__KEY
OA\$MRG__TTOUT function, 6-220
 to 6-221
 See also MERGE,
 MERGE__LINE
 and MERGE, 6-135
 and MERGE__LINE, 6-142
OA\$MSG__FAC
 special symbol, 14-25
OA\$MSG__ID
 special symbol, 14-25
OA\$MSG__PURGE function, 6-222
OA\$MSG__SEV
 special symbol, 14-25
OA\$MSG__TEXT
 special symbol, 14-25
OA\$MSG__TRAP function, 6-223
OA\$PAGE__LENGTH
 special symbol
 and MERGE, 6-136, 10-27
OA\$PAGE__NUMBER
 special symbol
 and MERGE, 6-136, 10-27
OA\$PROMPT__HELP
 special symbol, 14-3
 and %OA-I-LASTLINE, 6-62
 and HELP, 6-269
 and HELP function, 6-111
 and prompts, 6-269

OA\$PROMPT_TEXT
 special symbol
 and PROMPT, 6-267, 6-271

OA\$SCL_BOTTOM function
 and Gold Down Arrow, 4-34

OA\$SCL_DOWN function
 and the Down Arrow, 4-34

OA\$SCL_FIRST_PAGE function
 and Gold T, 4-34

OA\$SCL_LAST_PAGE function
 and Gold B, 4-34

OA\$SCL_NEXT_PAGE function
 and Gold TAB, 4-34

OA\$SCL_nnn functions, 6-224
 to 6-227
See also individual functions

OA\$SCL_PRIOR_PAGE function
 and Gold BACKSPACE, 4-34

OA\$SCL_TOP function
 and Gold Up Arrow, 4-34

OA\$SCL_UP function
 and the Up Arrow, 4-34

OA\$SCP_STACK_DUMP
 function, 6-228

OA\$SCRIPT_EXIT_HANDLER
 special symbol
 and Gold X, 15-4
 and Gold X, CTRL/C, 4-59

OA\$SCRIPT_PSIB
 script pseudo-input buffer, 15-3
 and script text lines, 15-3
 in script DO Mode, 15-4

OA\$SCRIPT_TERM
 script pseudo-input buffer
 terminator, 15-4

OA\$SCRIPT_TEXT
 script pseudo-input buffer
 text, 15-4

OA\$SEL_ADDR
 special symbol
 used with FOR function, 3-73

OA\$SEL_COUNT
 special symbol
 used with FOR function, 3-73

OA\$SEL_DSAB
 special symbol
 used with FOR function, 3-73

OA\$SEL_KEY
 special symbol
 and validation post-
 functions, 4-51
 used with FOR function, 3-74

OA\$SEL_LINE
 special symbol
 used with FOR function, 3-74

OA\$SHARE_CUR
 and CABINET
 SHARE_DOCUMENT, 9-47

OA\$SHARExxx logicals, 9-19
See also File Cabinet shared
 area

OA\$SRC_COUNT
 special symbol
 used with FOR function, 3-74

OA\$STAT_GET_STATISTICS
 function, 6-229 to 6-230

OA\$STATUS
 special symbol
 and CABINET, 6-6
 and CABINET
 NEXT_FOLDER, 9-36
 and CABINET subfunction
 calls, 9-15

OA\$SUB_CLOSE function, 6-231

OA\$SUB_OPEN function, 6-232

OA\$SYM_CLOSE function, 6-233

OA\$SYM_GET_SYMBOL
 function, 6-234 to 6-235

OA\$SYM_MAKE_SYMBOL
 function, 6-236 to 6-237

OA\$SYM_OPEN function, 6-238
 to 6-239

OA\$TABLE
 validation DSR, C-8

OA\$TRA_OUT function, 6-240

OA\$TRA_SET function, 6-241
 to 6-243

OA\$TRM_BRDCST_INIT
 function, 6-244 to 6-244.1

OA\$TRM_BRDCST_STOP
 function, 6-244.2 to 6-244.3
OA\$TRM_INIT function, 6-245
OA\$TRM_INQUIRE function,
 6-246
OA\$TRM_PORT_LINE
 function, 6-247
OA\$TRM_PORT_LOG_ON
 function, 6-248
OA\$TRM_PORT_OFF
 function, 6-249
OA\$TRM_PORT_ON function,
 6-250
OA\$TRM_PORT_PRINT
 function, 6-251
OA\$TXL_CLOSE function, 6-252
OA\$TXL_COMPILE function,
 6-253
OA\$TXL_DIRECTORY
 function, 6-254
OA\$TXL_INSTALL function,
 6-255
OA\$TXL_LIST function, 6-256
OA\$TXL_OPEN function, 6-257
 to 6-258
OA\$TXL_REMOVE function,
 6-259
OA\$TXL_SEARCH_LAST, 15-6
OA\$VAL_SET_VALID
 function, 4-50, 4-54, 6-260 to
 6-261
OA\$VIEW_FIELDS
 View Fields form, 4-63, 6-149,
 16-18
OA\$VIEW_MESSAGES
 and Gold H, 4-60
 and message buffer, 14-25
 View Messages form, 4-60,
 16-18
OA\$VIEW_ND
 View Named Data form, 4-60,
 6-149, 16-18
OA\$YESNO
 validation DSR, C-9
OA\$YN
 validation DSR, C-9

OAHELP
 logical name, 14-21
 main-process logical, 6-110
OAINI.COM
 BYE function redefined in, 6-24
 EXIT function in, 6-79
 subprocess log-in file, 7-2
OAMAILBOX
 and DCLMAILBOX, 7-4
 calling multiple functions
 through, 6-8
 mailbox from subprocess, 7-4
 multiple writes to, 7-7
 syntax while writing to, 7-6
OAMESS.MSG
 message file, 14-23
 updating the, 14-23
OAUSER
 and COMMAND function, 6-36
 and DCL function, 6-53
 and SET_DEFAULT, 6-299
 subprocess versus main
 process, 7-17
OAUSER:ACTITEM.DAT
 file structure, 3-40
OAUSER:CALACCESS.DAT
 Action Items file, 13-1
OAUSER:DAF.DAT
 See PDAF
OAUSER:DOCDB.DAT
 personal File Cabinet index,
 9-2, 9-4
 keys to, 9-4, 9-5
 reorganizing
 and CABINET
 REORG_DOCDB, 9-44
OAUSER:PASTE.TMP
 See Scratch Pad
/ONCE form qualifier
 in entry forms, 3-52
/ONCE menu qualifier, 3-28
/ONE_ENTRY entry form
 qualifier, 3-53
 and /SAVE_START, 3-53
/OPTIONAL field qualifier, 4-21,
 4-23 to 4-24, 4-49

/OVERLAY form qualifier, 3-20

P

Pad, scratch

and Gold *, 4-61

PARSE_USER function, 6-262 to 6-263

from the subprocess, 7-11

PDAF, 9-2, 9-9, 9-46

and OUSER:DOCDB.DAT, 9-9

Performance

and /HIGHLIGHT, 3-18

and FMS field highlights, 16-23

and RMS tuning, 16-23

and symbol types, 8-15, 16-23

command procedures versus scripts, 16-22

effect of stacking functions on, 6-8

Permanent symbol, 34

Permanent symbol table, 34
username.PST, 8-31

Permanent symbols, 8-2, 8-31 to 8-36

Personal Document Attributes File
See PDAF

Pipeline mechanism, 6-9
source functions

MERGE, 6-134, 6-141

with COPY, 6-44

PORT_OFF function, 6-264

PORT_ON function, 6-264

and PRINT_SCREEN, 4-61

/POST_FUNCTION

appended in FORM function calls, 3-9

/POST_FUNCTION field
qualifier, 4-24 to 4-25

and the prior menu stack, 3-75
versus /POST_FUNCTION

form qualifier, 4-24

/POST_FUNCTION form

qualifier, 3-20, 3-21

and the prior menu stack, 3-75

/PRE_FUNCTION

appended in FORM function calls, 3-9

/PRE_FUNCTION field

qualifier, 4-25 to 4-26

and the prior menu stack, 3-75

/PRE_FUNCTION form

qualifier, 3-21

and the prior menu stack, 3-75

Precompiling forms, 16-11

with the PCF option, 16-11

PREVIOUS PAGE key

Gold BACKSPACE, PREV
SCREEN, 4-61

PRINT SCREEN key

and PRINT_SCREEN
function, 4-61

Gold P, 4-61

PRINT_SCREEN

saving in PASTE.TMP, 4-61

PRINT_SCREEN function, 6-265 to 6-266

from the editor, 4-61

Gold P, 4-61

Printing files

See QUEUE_PRINT

Prior menu stack, 3-35

Profile

See User Profile

Profile, user

privileges

PRVCMD, 3-23, 3-32

PRVDCL, 3-32

PRVSUP, 3-23

Programming Functions, 16-12 to 16-14

/PROMPT field qualifier, 4-26 to 4-27

PROMPT function, 6-267 to 6-270
and OA\$FLD_nnn

functions, 4-67

and user-defined keys, 6-269

from the subprocess, 7-13

PROMPT_ID function, 6-271 to 6-272

and OA\$FLD_nnn
functions, 4-67

from the subprocess, 7-13

PRVCMDB

User Profile privilege, 3-23,
3-32, 6-2

PRVDCL

User Profile privilege, 3-32, 4-58

PRVSUP

User Profile privilege, 3-23

PURGE_FILE function, 6-273
to 6-276

PUT_FIELD function, 6-277 to 6-278

/PUT_SAVE field qualifier, 4-27
to 4-28

Q

Qualifiers

See also Field qualifiers, Form
qualifiers

Field versus Form, 3-10

Qualifiers, field

used to define a field, 4-2

QUEUE_BATCH function, 6-279
to 6-283

versus COMMAND, 6-282

QUEUE_PRINT function, 6-284
to 6-288

Quit form

See EXIT SCREEN

R

Recognition

full

and Gold S, 4-57

partial

and Gold L, 4-57

with /INVALID, 4-19

with /RECOGNITION, 4-28

with /VALID, 4-47

/RECOGNITION field qualifier,
4-28 to 4-30
with /VALID, 11-50

Record collection

controlling output of, 4-55 to 4-56

with /USE_FORM, 4-55

creating

with FOR function, 6-83
to 6-91

with select form, 3-68, 3-71

with validation qualifiers,
4-55

displaying

with SEL_DISPLAY
subfunction, 3-72, 6-87

formatting

with /SHOW, 4-45

with /STYLE select-form
qualifier, 3-70

with search form .SOUT
directive, 3-67

with SEL_STYLE
subfunction, 3-72, 6-87

number of records in

OA\$SEL_COUNT, 3-73

putting unique records into

with /UNIQUE, 4-47

refining

with FOR FIRST, 6-84

with FOR UNIQUE, 6-84

saving

with /LIST select-form
qualifier, 3-69

selecting from

with /STYLE select-form
qualifier, 3-70

with OA\$SEL_ADDR, 3-73

with OA\$SEL_KEY, 3-74

with OA\$SEL_LINE, 3-74

with SEL_CHOICE
subfunction, 3-72, 6-87

source of records in a

OA\$SEL_DSAB, 3-73

viewing

with Gold L, 4-57

with Gold S, 4-57

Record Management Services

See RMS

Record selection expression

See RSE

Redefining the keyboard, 4-63 to 4-65

Relational operators, 6-85

RENAME function, 6-289 to 6-290

REPEAT function, 6-291 to 6-292

in function lists, 6-7

/RESET form qualifier, 3-22

in FORM function calls, 3-9

using the, 3-77

RESULT

common DCL symbol, 7-12

and LOGICAL, 6-126

updating, 6-102

DCL symbol

set by OA\$FLD__DONE,

OA\$FLD__EXIT, 4-67

Retrieving symbol values

with /GET__SAVE, 4-16

RETURN key

and OA\$FLD__DONE, 4-58

Right Brace character

used as script mnemonic

delimiter, 15-3

RMS indexed files

defined by entry forms, 3-37

RMS key of reference

See also Data file keys

in an FDL-created file, 3-39

RMS tuning

for improved performance,

16-23

RSE

differences between search and select forms, 3-61

in field validation, 4-30 to 4-32

in search form .TYPE entry, 3-63

in the FOR function, 6-83

/RSE__INVALID field qualifier,

4-4, 4-30 to 4-32

/RSE__RECOG field qualifier,

4-30 to 4-32

/RSE__VALID field qualifier,

4-4, 4-30 to 4-32

S

SAVE function, 6-293 to 6-295

from the subprocess, 7-11

/SAVE__START entry form

qualifier, 3-53

and /KEY key fields, 3-53

and /ONE__ENTRY, 3-53

Scratch Pad

and Gold*, 4-61

SCRIPT function, 6-296 to 6-297

See also DO

Script processing

uses of scripts, 15-1

Scripts, 15-1 to 15-34

See also SCRIPT, DO functions

adding comments to, 15-25

block output

beginning, 15-10

calling functions from, 15-14,

15-15

conditional processing in, 15-17

controlling cursor location

in, 15-12, 15-13, 15-26

controlling screen appearance

in, 15-11, 15-12, 15-24, 15-26,

15-30

debugging, 15-29

directives

.ABORT, 15-4, 15-9

as functions, 15-31

.BLOCK, 15-10

.CHECK__FIELD, 15-10

.CHECK__FORM, 15-11

.CLEAR, 15-11

.CUR__KEEP, 15-12

.CUR__RESTORE, 15-13

.CUR__SAVE, 15-13

.CURSOR, 15-12

.DISPLAY, 15-13

.EXIT, 15-4, 15-14

.FUNCTION, 15-14

Scripts (Cont.)

- .FX, 15-15
- .FZ, 15-15
- .GET_FIELD, 15-16
- .GOTO, 15-17
- .IF, 15-17
- .JUDGE, 15-19
- .LABEL, 15-21
- .PAUSE, 15-21
- .PROCESS, 15-22
 - and
 - OA\$SCRIPT_PSIB, 15-3
- .PROMPT, 15-24, 15-28
- .REFRESH, 15-24
- .REMARKS, 15-25
- .RESET_CURSOR, 15-26
- .SET_PAUSE, 15-26
- .SET_REFRESH, 15-26
- .SHOW_FORM, 15-27
- .SINX, 15-28
- .TEXT, 15-29
- .TRACE, 15-29
- .TRACE_ON, 16-15
- .VIDEO_ATT, 15-30
- .WAIT, 15-30
- displaying forms in, 15-27
- example, 15-32
- exiting from, 4-59, 15-4, 15-9, 15-14
- field processing in, 15-16
- flow control in, 15-17, 15-19, 15-21
- format, 15-7
- graphics in, 15-13
- interactive directives
 - and OA\$SCRIPT_PSIB, 15-3
- mnemonic delimiters, 15-3
- mnemonics for input, D-1
- nested, 15-2
- output from, 15-29
- pausing in, 15-21, 15-26
- processing input while in, 15-22
- prompting from, 15-24, 15-28, 15-30

- synchronizing, 15-10, 15-11
 - text lines, 15-8
 - terminators in, 15-8
- tracing
 - with Gold (, 6-34
 - two modes of, 15-2
- DO, 15-2
- mixing the, 15-2

Scripts mnemonic, 15-3

Scripts, command procedures

- versus performance in

- applications, 16-22

/SCROLL field qualifier, 4-33 to 4-44, 6-224

Scrolled region

- creating with FMS, 4-34

- identifying with /SCROLL, 4-33

- navigating in

- with OA\$SCL_nnn

- functions, 6-224

Scrolling

- display-only, 4-43

Scrolling functions

- See OA\$SCL_nnn

SDAF, 9-2, 9-9, 9-46

- and OAUSER:DOCDB.DAT, 9-9

Search form, 3-60 to 3-68

- and FOR function, 3-65

- Named Data, 3-61

- .SFILE directive, 3-65

- .SOUT directive, 3-67

- processing, 3-68

- qualifiers, 3-65

- .TYPE entry syntax, 3-60, 3-63 to 3-65

- differences between select

- and search, 3-61

- signaling errors in, 3-65

- .TYPE SEARCH, 3-6

SEARCH key

- Gold S, 4-57

SEARCH132 arg form

- and record collection output, 3-67

- Searching text files
 - with search form .SFILE directive, 3-65
- SEARCHKEY arg form
 - and record collection output, 3-67
- Section
 - globally shared, 16-11
- Segmented form
 - See Form set
- SEL_CHOICE
 - FOR subfunction
 - and OA\$SEL_LINE special symbol, 3-74
 - in select forms, 3-72
 - special FOR subfunction, 6-87
- SEL_DISPLAY
 - FOR subfunction
 - and /SHOW, 4-45
 - in select forms, 3-72
 - special FOR subfunction, 6-87
- SEL_STYLE
 - FOR subfunction
 - and /LIST, 3-70
 - in select forms, 3-72
 - special FOR subfunction, 6-87
- Select form, 3-68 to 3-74
 - Named Data, 3-69
 - processing, 3-72
 - summary, 3-74
 - .TYPE line, 3-71
 - qualifiers, 3-69
 - /LIST, 3-69
 - /STYLE, 3-70
 - required features, 3-69
 - .TYPE SELECT, 3-6
- Selection List
 - and NEXT_LIST, 6-155 to 6-156
 - created by /LIST select-form qualifier, 3-70
 - created by search form, 3-60
 - identifying
 - with /LIST field qualifier, 4-22

- identifying next selection in, 4-23
- opening and reading
 - with
 - OA\$FLD_NEXT_LIST, 4-60
- Sequential files
 - See Text files
- SET_DEFAULT function, 6-298
 - to 6-299
 - and COMMAND function, 6-36
 - and DCL function, 6-53
 - and NEWDIR, 6-150
 - and OAUSER, 6-299
- SET_MENU function, 3-32, 6-300
 - to 6-301
- .SFILE directive
 - in search form Named Data, 3-65
 - syntax, 3-66
- Shared Document Attributes File
 - See SDAF
- Shared libraries
 - in the File Cabinet, 9-19 to 9-20
- /SHOW field qualifier, 4-45 to 4-46, 4-55
 - and SEL_DISPLAY, 4-45
 - with multiple data sets, 4-45
- SHOW_FORM function, 3-4, 3-12, 6-302
 - and nonstandard forms, 3-80
- SORT_LIST function, 6-303 to 6-305
 - See also MERGE_LIST, TEST_SPEC
- .SOUT directive
 - in search form Named Data, 3-67
 - syntax, 3-67
- SPAWN function, 6-306 to 6-307
 - and the primary subprocess, 7-19
- Special symbol
 - table, 41

- Special symbols, 8-3, 8-16 to 8-30
 - DTR\$, 6-66
 - performance versus DSRs, 16-23
 - read-only, 8-15 to 8-30
 - read-write, 8-15 to 8-30
 - STAT\$, 6-308
 - used with FOR function, 3-73
 - write-only, 8-15 to 8-30
- Special-function keys
 - See* Keyboard directives
- Stacking functions, 6-4 to 6-9
 - See also* Function lists
- Stacking menu options, 3-32
 - restrictions when, 3-33
- Statistics
 - ALL-IN-1 release
 - and VERSION, 6-316
 - data set
 - and OA\$DSA__CACHE__STATUS, 6-169 to 6-170
 - form library
 - and OA\$FBT__LIST__LIBTREE, 6-175 to 6-176
 - and OA\$FBT__LIST__TREE, 6-177 to 6-178
 - process
 - and STATISTICS, 6-308 to 6-309
- STATISTICS function, 6-308 to 6-309
- /STORE field qualifier, 4-46
- /STYLE select form qualifier
 - and /LIST, 3-71
 - and SEL__STYLE
 - subfunction, 3-72
 - used to format record collection, 3-70
- Subprocess, 7-1 to 7-23
 - accessing with DCL function, 6-51 to 6-53
 - calling function lists from, 6-8
 - creating an alternate with SPAWN, 6-306
 - initializing, 7-1
 - obtaining user input while in, 7-20 to 7-23
 - passing data to programs in, 7-20
 - receiving data from through OAMAILBOX, 7-4
 - running command procedures in, 7-19
 - sending data to through DCLMAILBOX, 7-4
 - sending text files to with INCLUDE, 6-115
 - trapping errors while in with ERROR, 6-6, 6-76 to 6-77
- Subsystem
 - defined, 1-6
- Subsystems, 3-24
 - interaction of, 1-6
- /SUPER form qualifier, 3-23
- Support, terminal, 8-2
- Symbol table
 - local, 8-4
 - permanent, 8-2
 - opened during initialization, 2-3
- Symbols, 8-1 to 8-36
 - basic types, 8-2 to 8-5
 - DSRs, 8-4
 - field name, 8-2
 - local, 8-4
 - permanent, 8-2
 - quoted string, 8-2
 - special, 8-3
 - concatenating in lists, 8-11
 - creating, 8-14
 - with /COPY field qualifier, 4-8
 - dates and times in, 8-13
 - See also* Date/time formats
 - defining and using in the subprocess, 7-8 to 7-16

Symbols (Cont.)

- getting values from, 8-15
 - with /GET_SAVE, 4-16
 - with GET, 6-101 to 6-107
- in function calls, 6-12
- in the main process, 8-15
- in the subprocess, 8-15
- iterative substitution, 8-12
- performance differences, 8-15
- putting values into, 8-14
 - with /PUT_SAVE, 4-27
- substrings, 8-10
- values stored in multiple, 8-15
- zero-suppressing numeric, 8-11

Symbols, DCL

- creating
 - with GET, 6-103

SY\$INPUT

- mailbox to subprocess, 7-4

T

TAB

- and OA\$FLD_NEXT, 4-62

Terminal Modes

- See* Modes

Terminal Support, B-2, B-5

TERMINATOR

- common DCL symbol, 7-13
- and interactive functions, 7-13
- DCL symbol
 - and PROMPT, 6-267, 6-271
 - set by OA\$FLD_DONE, OA\$FLD_EXIT, 4-67

Terminator functions

- See* OA\$FLD_nnn
- internal, 4-66

Terminator key(s)

- processing, 4-66 to 4-69
- in menu choice field, 3-31

TEST_SPEC function, 6-310 to 6-311

- See also* MERGE_LIST, SORT_LIST

Text data sets, 10-2

categories

- ASCII, 10-2
- COMPOUND, 10-2
- DX, 10-2
- DXT, 10-2
- MAIL_ATTACHMENT, 10-2
- MAIL_HEAD, 10-2
- PIPELINE, 10-3
- WPL, 10-3
- OA\$EDIT_BUFFER, 6-75
- writing to
 - with DESTINATION, 6-58

Text files

- copying
 - with COPY, 6-43 to 6-44
- creating
 - with EDIT, 6-71 to 6-73, 10-3
 - with MERGE, 6-134 to 6-140
 - with MERGE_LINE, 6-141 to 6-142
 - with MERGE_LIST, 6-143 to 6-146
- deleting, 6-56 to 6-57
- displaying
 - See also* OA\$CNV_FROM_SCROLL
 - See also* OA\$CNV_TO_SCROLL
 - with LIST, 6-120
- extensions and editors, 10-3
- formatting, 10-28 to 10-33
- including text into
 - with INCLUDE, 6-114 to 6-115
- internal structure versus editor style, 10-21
- modifying
 - See also* OA\$CNV_FROM_SCROLL
 - See also* OA\$CNV_TO_SCROLL
 - with EDIT, 6-71 to 6-73
 - (default editor), 10-4 to 10-19

Text files (Cont.)

- naming
 - with /DOC, 4-11
- printing
 - with QUEUE_PRINT
 - on an attached printer, 6-264
- processing, 10-1 to 10-33
- searching
 - with search form .SFILE
 - directive, 3-65
- validating existence of
 - with /FILE_CHECK, 4-14
- WPS-PLUS
 - and MERGE_LIST, 6-143
 - to 6-146
 - formatting
 - with FORMAT, 6-97 to 6-100

Text processing, 10-1 to 10-33
editor menu, 6-74 to 6-75

Time

- formatting
 - See Dates and times

TIME function, 6-312 to 6-313

Time Management, 13-1 to 13-30

- activities, 13-2
- CALENDAR, 6-26
- organization, 13-1

/TITLE form qualifier, 3-23

TRACE keys, VERIFY and
Gold (,Gold), 4-62

TXL

- and the OAGBL.BLI
 - module, 15-5
 - building and modifying, 15-5
 - search order, 15-6
 - changing with
 - OA\$TXL_SEARCH_LAST,
 - 15-6
 - to increase performance, 15-5
- .TYPE or type
See Form .TYPE

U

/UNIQUE field qualifier, 4-47, 4-52

UNWIND function, 6-314 to 6-315

Up Arrow

- and OA\$FLD_UP, 4-62
- calls OA\$SCL_UP in scrolled
- region, 4-34

/USE_FORM field qualifier, 4-47

- /USE_FORM validation
 - subqualifier, 4-21, 4-49, 4-51, 4-55
 - with /INVALID, 4-19
 - with /RECOGNITION, 4-28
 - with /RSE_nnn, 4-31

/USER form qualifier, 3-24

User Profile

- ACTITEM field, 6-18
- and NEWLIB, 6-152 to 6-154
- DOCFRM field
 - and NEWDIR, 6-150
- Form Libraries field (FRMLIB)
 - use of, 3-3
- FRMLIB field, 16-9
- opened during initialization, 2-2
- privileges
 - PRVCMD, 3-23, 3-32, 6-2
 - PRVDCL, 3-32, 4-58
 - PRVSUP, 3-23

User-defined functions

- See Functions, user-defined

User-defined keys

- See Keyboard directives

V

/VALID field qualifier, 4-4, 4-47
to 4-50

- and /RECOGNITION, 4-49
- on entry form key fields, 3-45
- with /RECOGNITION, 11-50

Validation

- and OA\$VAL_SET_VALID,
- 6-260

field qualifiers, 4-4

- /AUTO, 4-5
- combining, 4-54
- /FILE_CHECK, 4-14 to 4-16
- /INVALID, 4-19 to 4-22
- /OPTIONAL, 4-23 to 4-24

Validation

field qualifiers (Cont.)

/RECOGNITION, 4-28 to 4-30

/RSE__INVALID, 4-30 to 4-32

/RSE__RECOG, 4-30 to 4-32

/RSE__VALID, 4-30 to 4-32

/UNIQUE, 4-47

/VALID, 4-47 to 4-50

optional

with /OPTIONAL, 4-23

with /RECOGNITION, 4-28

through field qualifiers, 4-51 to 4-56

with RSEs, 4-30 to 4-32

Validation expression, 4-51, 45

DSR in a, 4-20, 4-29, 4-48, 4-51

with /INVALID, 4-19

with /RECOGNITION, 4-28

with /VALID, 4-47

VAX Information Architecture

and ALL-IN-1, 1-4

overview, 1-3

VERIFY and TRACE keys

Gold (,Gold), 4-62

Verifying

command procedures

with Gold (, 6-34

Version 2.X, converting to

form libraries, 16-9

VERSION function, 6-316 to 6-317

View form, 16-18

See also individual forms

.TYPE VIEW, 3-6

VMS

file names

See File names

W

WAIT function, 6-270, 6-318 to 6-320

and OA\$FLD__nnn

functions, 4-67

from the subprocess, 7-13

WPINDEX

select form

processing of, 3-72

WPS-PLUS

and MERGE__LIST, 6-143 to 6-146

and SORT__LIST, 6-303 to 6-305

and TEST__SPEC, 6-310 to 6-311

WPSPLUS\$CBI__CLOSE__WINDOW, 6-321

WPSPLUS\$CBI__CLOSE__WINDOW2 function, 6-322

WPSPLUS\$CBI__OPEN__WINDOW function, 6-323 to 6-324

WPSPLUS\$CBI__OPEN__WINDOW2 function, 6-325 to 6-326

WPSPLUS__SPELL function, 6-327

WRITE function, 6-328 to 6-329

and entry-form modes, 3-78

called by FOR, 6-91

Y

YESNO__PROMPT function, 6-270, 6-330 to 6-332

See also PROMPT

and OA\$FLD__nnn

functions, 4-67

from the subprocess, 7-13

YESNO__PROMPT__ID

function, 6-333 to 6-334

See also PROMPT

and OA\$FLD__nnn

functions, 4-67

from the subprocess, 7-13

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
and Puerto Rico
call **800-258-1710**

In Canada
call **800-267-6146**

In New Hampshire,
Alaska or Hawaii
call **603-884-6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: P&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809-754-7575

Reader's Comments

Your comments and suggestions are welcome. They will help us in our continuous effort to improve the quality and usefulness of our publications.

1 How would you rate this manual for:

	good	fair	poor
completeness of information	_____	_____	_____
accuracy of information	_____	_____	_____
ease of use (clarity, organization)	_____	_____	_____

2 Did you find errors in this manual? Please specify by page and paragraph, or mark corrections on pages and attach copies to this form.

	page	paragraph
a) Incorrect information:	_____	_____
b) Information left out:	_____	_____
c) Hard to understand:	_____	_____

3 What suggestions do you have for improving this manual?

4 What parts of the manual were especially good at helping you understand things?

5 Please check the boxes that apply to you.

- | | |
|---|--|
| ☐ Technical | ☐ Used other word processors |
| ☐ Nontechnical | ☐ Used computers |
| ☐ Management | ☐ Use the word processing unit yourself |
| ☐ Nonmanagement | |
| ☐ Other (explain)_____ | |

Name_____Title_____

Company_____

Street_____City_____

State_____Zip_____Date_____

-----Do Not Tear - Fold Here and Tape-----

digital

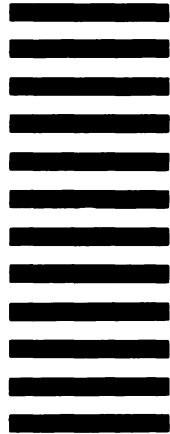


No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Manager OFFICE PROGRAM ZK02-1/N20
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, N.H. 03062



-----Do Not Tear - Fold Here and Tape-----

Cut Along Dotted Line



OFFICE MENU

Application Programmer's Reference
Volume 1: Flow Control

Update Package #1, January 1986

This package updates the ALL-IN-1 Office Menu Application Programmer's Reference Volume 1 (Order No AA-N324B-TE) for ALL-IN-1 Version 2.1.

In this package, a bar (|) in the outside margin shows where information has been changed or added. A bullet (●) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Insert the pages as follows.

Insert page	In place of
Guide to ALL-IN-1 Information	Guide to ALL-IN-1 Information
iii - xlii	iii - xlii
xlvi - 1	xlvi - 1
3-37 - 3-46	3-37 - 3-46
3-53 - 3-56	3-53 - 3-56
3-71 - 3-72	3-71 - 3-72
3-79 - 3-80	3-79 - 3-80
4-9 - 4-10	4-9 - 4-10
4-25 - 4-26	4-25 - 4-26
4-55 - 4-56	4-55 - 4-56
Index-1 - 36	Index-1 - 38



OFFICE MENU

Application Programmer's Reference
Volume 1: Flow Control

Update Package #2, April 1987

Update the Application Programmer's Reference Volume 1 (AA-N324B-TE) with this package. You must first have updated the APR with update package #1, January 1986 (AD-N324B-T1). When you have inserted both packages, the APR will be updated for ALL-IN-1 Version 2.2.

In this package, a bar (|) in the outside margin shows where information has been changed or added. A bullet (●) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Insert the pages as follows.

Insert page	In place of
xxi – xxii	xxi – xxii
3-21 – 3-22	3-21 – 3-22
Index-15 – 16	Index-15 – 16
Index-21 – 22	Index-21 – 22
Index-25 – 28	Index-25 – 28

AA-N324B-TE
Printed in U S A